Universidad Carlos III de Madrid
Departamento de Informática

Tesis Doctoral

Clasificación mediante Enjambre de Prototipos

Autor: Alejandro Cervantes Rovira

Directores:
Pedro Isasi Viñuela, Inés Mª Galván

Leganés, 2009

Índice general

Índice de figuras

Índice de tablas

1 Introducción

1.1 Ámbito de la Tesis

Se define la "Búsqueda de Conocimiento" como el proceso de identificar esquemas en un conjunto de datos que sean válidos, novedosos, potencialmente útiles y comprensibles. Esta definición extiende el concepto de minería de datos para incorporar las fases de selección, preproceso, minería propiamente dicha, interpretación y evaluación. Dado el crecimiento en el volumen de datos que las aplicaciones modernas deben procesar, se realiza mucha investigación en la forma en que pueden escalar los algoritmos de minería, así como el proceso complementario de selección de datos relevantes entre los datos disponibles.

En nuestro trabajo nos centraremos en tareas de clasificación. Un clasificador es cualquier sistema capaz de predecir la clase que debe asignarse a un conjunto de datos agrupados, que denominamos "patrón". Cada patrón está definido por los valores de sus "atributos". El proceso por el que un clasificador es capaz de predecir dicha clase puede basarse en información que se le proporciona a priori (reglas introducidas por un experto) pero, en el campo de la Inteligencia Artificial, se suele enfocar la investigación hacia sistemas que son capaces de aprender la relación entre atributos y clases por sí solos.

Para hacer referencia a un ejemplo sencillo, los datos de la historia clínica de un paciente podrían constituir un patrón; los atributos del mismo incluirían información identificada por etiquetas (sexo, nacionalidad), o bien por datos numéricos (edad, peso, valores de indicadores de pruebas médicas). En este ejemplo, el diagnóstico del paciente en relación con una enfermedad podría abstraerse como un problema de clasificación: la clase sería binaria

si padece la enfermedad (sí o no). Otros problemas que se podrían definir son su grado de propensión a una dolencia específica. En cualquier caso, la "clase" que el sistema predice suele definirse como un elemento dentro de un conjunto de etiquetas predefinido.

Cuando el clasificador dispone de un conjunto de patrones modelo, para los cuales conoce la clase asociada, se pueden aplicar modelos de "aprendizaje inductivo supervisado". En estos, el conjunto de datos conocido (patrones de aprendizaje) se analiza para generar un modelo de clasificación. Realizada esta tarea, el clasificador debe ser capaz de generalizar la información disponible para predecir la clase que corresponde a cualquier otro patrón de datos que se le presente. La razón por la que el clasificador puede realizar esta predicción es por la correcta representación de las correlaciones que existen entre los atributos y la clase asociada a cada patrón.

Un tipo concreto de clasificador es el que se basa en la regla del vecino más próximo. Este clasificador se considera de tipo "perezoso" (o "vago"), porque los datos de entrada no se preprocesan de ninguna forma. Todos los patrones conocidos por el clasificador se retienen; cuando el clasificador debe predecir la clase que se debe asociar a un patrón desconocido, simplemente se le asigna la clase del patrón conocido más cercano. Dicha cercanía está definida en términos de una función de proximidad (normalmente, una función distancia) que debe definirse previamente.

Una forma simple de representación de un clasificador de este tipo consiste en suponer un problema en el que existen dos atributos numéricos. En este caso, el espacio de atributos será una sección del plano. Sobre este espacio puede utilizarse simplemente la distancia Euclídea para medir la proximidad entre los patrones. Se observa que cada patrón conocido define a su alrededor una región en la que es el responsable de atribuir el valor de la clase; la unión de dichas regiones constituye la teselación de Voronoi de espacio de atributos. En el caso de que la distancia sea la Euclídea, las regiones de Voronoi tienen bordes rectos.

El cálculo de distancias entre patrones es un proceso costoso; por ello, en estos casos se procuran utilizar métodos de selección de datos para reducir el coste computacional del proceso de clasificación. Una vez realizada la selección de datos, el clasificador retiene una cantidad de información mucho menor y por consiguiente el proceso de clasificación se acelera. La información que se retiene es un conjunto de "prototipos". Hay varias formas de generar estos conjuntos de prototipos: algunos sistemas se limitan a escoger como prototipos algunos de los patrones (Selección de Prototipos);

otros, escogen los prototipos sin necesidad de que coincidan con los patrones conocidos (Reemplazo de Prototipos).

Ambos problemas se pueden abordar mediante algoritmos de búsqueda metaheurística, como los Algoritmos Evolutivos o los llamados Algoritmos de Inteligencia de Enjambre. La ventaja de una aproximación de este tipo es que el algoritmo se comporta de forma robusta y eficiente en un conjunto amplio de dominios. Uno de estos últimos algoritmos se denomina Optimización mediante Enjambre de Partículas (Particle Swarm Optimization o PSO). El algoritmo PSO es muy utilizado para la resolución de problemas de optimización, y se ha popularizado gracias a su rapidez de convergencia y la simplicidad de su implementación. Sin embargo, aún no se utiliza de forma sistemática en problemas de clasificación.

La tesis se enmarca en el ámbito de la clasificación mediante vecino más próximo utilizando Enjambres de Partículas. Para ello se propondrá un enfoque nuevo para PSO que permite abordar de forma práctica el problema de reemplazo de prototipos para clasificación. Denominaremos a este nuevo algoritmo Enjambre de Prototipos. Esta aportación puede generalizarse a otros campos de aplicación que resulten difíciles de abordar mediante el algoritmo PSO convencional. En este sentido abre un campo de aplicación novedoso para la comunidad de investigación en Inteligencia de Enjambre.

1.2 Objetivos de la Tesis

Se podría utilizar directamente el algoritmo PSO en problemas de clasificación. Para ello, habría que transformar primero el problema de clasificación en un problema de optimización, mediante una codificación adecuada de las soluciones en las partículas del enjambre. Sin embargo, esta forma de resolver el problema tiene inconvenientes importantes:

- La dimensión del espacio de búsqueda crece de forma proporcional al número de prototipos que se desea obtener.
- En PSO la dimensión de las partículas es fija e igual para todas ellas, de forma que se carece de flexibilidad para que el algoritmo escoja por sí solo los valores más adecuados para el problema de clasificación que se desea resolver.
- Existe un problema de simetrías en la codificación: una partícula representaría un conjunto de prototipos en un orden determinado; sin

embargo todas las permutaciones de dicho orden corresponden realmente con una única solución al problema. Es conocido que este tipo de situaciones perjudican el rendimiento de cualquier metaheurística de búsqueda.

En consecuencia, los objetivos para la presente Tesis Doctoral se pueden contemplar desde dos puntos de vista:

- **Extensión del campo de aplicación** del paradigma de PSO a los problemas de clasificación, de forma que se eviten los problemas de falta de escalabilidad, flexibilidad y simetrías en la codificación antes mencionados.
- **Obtención de un algoritmo competitivo** en el problema de reemplazo de prototipos. La calidad de dicho algoritmo se medirá en términos de:
  - Un buen resultado en términos de capacidad de generalización, es decir, la aplicación del algoritmo a patrones no conocidos (predicción). Esta medida es el objetivo básico de todo sistema de clasificación
  - Un conjunto reducido de prototipos en la solución. De este modo, la predicción podrá hacerse con un coste computacional mínimo.
  - Coste computacional del entrenamiento asumible. El cálculo de distancias a prototipos es un proceso computacionalmente intensivo, y un sistema de tipo inductivo tiene que realizar este proceso de forma repetida, durante un número de iteraciones y para una población de individuos. Por lo tanto, debe prestarse atención a este aspecto.
  - Aplicabilidad a conjuntos diversos de datos, incluidos conjuntos de datos con patrones ruidosos.

1.3 Evaluación de la Tesis

En los capítulos correspondientes de este trabajo se realiza una evaluación experimental del algoritmo propuesto. Para realizarla, se han seleccionado los siguientes elementos:

- Un conjunto de algoritmos representativos del estado del arte en problemas de clasificación
- Una implementación propia de un sistema de reemplazo de prototipos basado en PSO estándar; haremos en este caso un énfasis especial en las diferencias entre el algoritmo que proponemos y este enfoque convencional.
- Un conjunto de dominios con datos generados de forma artificial, que usaremos para verificar que el comportamiento del algoritmo corresponde con sus objetivos de diseño.
- Un conjunto de dominios (conjuntos de patrones), extraídos de colecciones de datos utilizadas comúnmente en la literatura para la evaluación y caracterización de algoritmos de clasificación

Un elemento importante en la evaluación de un nuevo algoritmo es la caracterización del mismo en función del tipo de problemas para los que es adecuado. Se extraerán conclusiones a este respecto a partir de la experimentación realizada.

Las condiciones del análisis experimental realizado se detallan en más profundidad en el Capítulo 4.

1.4 Esquema del Documento

El documento está estructurado como detallamos a continuación.

En el Capítulo 2 muestra el estado del Arte relevante para la fundamentación y desarrollo del algoritmo. Se hará especial énfasis en dos campos concretos: clasificación mediante vecino más próximo, y optimización mediante enjambre de partículas (PSO). Se tratarán con más profundidad los aspectos y aplicaciones que consideramos de más relevancia para la justificación y comprensión de la aportación realizada.

En el Capítulo 3 se describe, por una parte, el algoritmo de clasificación basado en PSO estándar que utilizamos como una de las referencias de comparación. A continuación se describe nuestra propuesta, el algoritmo PSC; en dicha descripción se hace referencia a las modificaciones que realizamos sobre la codificación y ecuaciones del apartado anterior. En este mismo capítulo se describen las variantes del algoritmo, que son objeto más

adelante de estudios experimentales en capítulos independientes. Dichas variantes se introdujeron para abordar aspectos específicos hallados o ideados con posterioridad a la experimentación de las versiones iniciales de PSC.

En el Capítulo 4 se detalla la metodología con la que se ha realizado la experimentación, los algoritmos y dominios a los que se refieren los resultados de la misma, y se especifica el método de obtención de resultados así como los parámetros del test de significación estadística utilizado en las tablas comparativas.

Los tres capítulos siguientes recogen los estudios experimentales: en el Capítulo 5 se detalla la experimentación realizada con el algoritmo PSC y la versión con población adaptativa (APSC); el Capítulo 6 realiza un estudio de tolerancia al ruido de APSC; y el Capítulo 7 describe la experimentación realizada con la variante de PSC y APSC con optimización de la medida de proximidad.

Por último, el Capítulo 8 expone nuestras conclusiones sobre el estudio realizado, y las propuestas de desarrollo del mismo en futuras líneas de trabajo.

En los Apéndices hemos recogido algunas tablas que pueden servir como referencia adicional, y que preferimos no incorporar al texto principal por claridad y legibilidad del mismo.

2

Estado del Arte

Hemos dividido el Estado del Arte relevante para nuestro trabajo en dos grandes secciones.

La primera se centra en el problema de Clasificación. En primer lugar se realiza una introducción general al problema, que incluye una breve taxonomía de las distintas familias y paradigmas de clasificación. En segundo lugar, nos centramos en los Clasificadores por vecino más próximo, y más específicamente los Clasificadores por Reemplazo de Prototipos, familia a la que pertenece el algoritmo que proponemos en este trabajo.

La segunda se centra en el algoritmo de Optimización mediante Enjambre de Partículas (PSO). Describe los principios de dicho algoritmo, así como algunas de sus variantes más importantes, y termina analizando algunas de las aplicaciones del mismo, especialmente las que abordan problemas de clasificación.

2.1 Métodos de Clasificación

2.1.1. Consideraciones generales

Un clasificador es un sistema capaz de asociar una etiqueta de "clase" a un conjunto de datos que denominamos "patrón". La información que permite atribuir una misma clase a varios patrones procede de las relaciones entre los datos que lo componen, denominados "atributos". Los atributos de los patrones pueden tomar la forma de etiquetas, valores binarios o numéricos.

Las relaciones o reglas que permiten asociar la clase al patrón pueden obtenerse mediante sistemas de aprendizaje automático. Se dice que se trata

de "aprendizaje supervisado" cuando existe una serie de datos de ejemplo o "patrones de entrenamiento" (E), etiquetados con la clase a la que pertenece, que puede proporcionarse al clasificador para que disponga de información en los que basar la forma en que realizará la predicción a patrones desconocidos o "patrones de validación" (V).

Una segunda forma de interpretar un clasificador supone que, en el proceso de análisis de la información disponible, se genera un "'conocimiento" sobre un problema determinado. Este conocimiento se genera mediante abstracción, reconociendo en los patrones ciertas regularidades o correlaciones que permiten expresar la relación entre atributos y clases.

No todas las aplicaciones de clasificación hacen accesible dicho conocimiento a un observador externo: algunas funcionan como "cajas negras" en las que no resulta posible realizar este tipo de aprovechamiento. Algunos ejemplos de clasificadores de este tipo son las redes de neuronas artificiales o las máquinas de soporte vectorial (Support Vector Machines o SVM, [Boser et al., 1992]).

En contraste con los anteriores, muchos clasificadores permiten expresar el proceso de clasificación mediante reglas que se pueden describir en términos de lógica formal de un tipo u otro. Algunas alternativas que permitirían agrupar los clasificadores por el "lenguaje" en el que se expresan serían:

- Reglas basadas en predicados, que toman la fórmula de cláusulas de lógica de primer orden sobre los valores de los atributos de los patrones. Puede tratarse de reglas conjuntivas simples ("Si $Atributo_1 = valor_1$ entonces $Clase = C_i$" o pueden usarse reglas de asociación que establecen relaciones entre los atributos ("Si $Atributo_1 < Atributo_2$ entonces $Clase = C_i$"). Entre ellos podemos citar RIPPER [Cohen, 1995], PRISM [Cendrowska, 1987], PART [Frank and Witten, 1998], C4.5 [Quinlan, 1993] o GAssist [Bacardit and Garrell, 2007].

- Reglas basadas en prototipos, que toman a forma de cláusulas de lógica sobre la similitud, expresada en términos de una función de proximidad δ (puede o no ser una distancia matemática) , entre los atributos de los patrones y los de un conjunto de ejemplos cuya clase conocemos y que sirve de referencia. Suelen denominarse también "Clasificadores por Similitud". En particular pertenecen a esta clase los clasificadores que usan la regla del prototipo más próximo: "Si $p_i = argmin_P(\delta(Patron_j, p_k \in P)), \text{entonces} Clase(Patron_j) = Clase(p_i)$". Entre ellos podemos citar IBK [Aha et al., 1991], o tam-

bién K* [Cleary and Trigg, 1995]. También podríamos incluir en esta categoría ciertos clasificadores basados en redes de neuronas como Learning Vector Quantization (LVQ, [Kohonen, 1995]) o Redes de Neuronas de Función de Base Radial (RBFNN, [Powell, 1987]).

- Reglas basadas en lógica borrosa, análogas a las primeras, pero expresadas en cláusulas que analizan la pertenencia de los atributos a conjuntos borrosos cuyas reglas de inferencia proceden del campo de la Lógica Borrosa. Son comunes en este campo las aproximaciones de tipo evolutivo, , como la propuesta en [Ishibuchi et al., 1999], donde se evolucionan conjuntos individuales de reglas o [Shi et al., 1999], en el que se codifica todo un sistema de clasificación borroso mediante un algoritmo genético.

Distintos tipos de regla pueden expresar relaciones de distinta complejidad entre los atributos. Sobre un "espacio de atributos" definido por el número de atributos, los patrones de la misma clase formarían regiones cuya separación imaginaria vendría dada por la "frontera de decisión". En [Duch and Grudzinski, 2001] se analizan los distintos tipos de reglas en función del tipo de frontera de decisión que son capaces de generar. En función del tipo de frontera (por ejemplo, si son lineales), se pueden definir clases de equivalencia entre los tipos de regla.

La forma de evaluar la utilidad de un sistema de clasificación es múltiple:

- Medidas cuantitativas sobre la predicción que realiza el clasificador. Podemos distinguir su capacidad de **representación**, o cómo las reglas generadas reproducen con exactitud la información disponible para el entrenamiento; y **generalización**, o cómo las reglas generadas predicen la clase de patrones nuevos.
- Comprensibilidad de las reglas, o si se puede extraer conocimiento útil de las mismas. Para evaluar la comprensibilidad, puede medirse el número de reglas, de tests sobre los atributos o de prototipos en el clasificador. Al estar relacionado con la representación del clasificador, estas medidas tendrán más utilidad al comparar entre sí clasificadores basados en la misma representación (reglas, prototipos, redes de neuronas, etc.).
- Interés para el usuario. La evaluación del interés del usuario requiere mayor información sobre la aplicación o el usuario concreto, y por lo tanto no suele analizarse de forma general.

Para realizar medidas numéricas de la **capacidad de representación** y de **generalización** se suele utilizar como referencia un sistema de clasificación con dos clases. Suele denominarse a las clases P (Positiva) y N (Negativa). En dicho clasificador se puede calcular la denominada "matriz de confusión" (Tabla 2.1), que se usa para comparar el número de patrones que el clasificador predice para cada una de las clases, con el valor esperado (real). En la tabla, los "True Positives" (TP) y "True Negatives" (TN) son patrones correctamente clasificados, mientras que los "False Positives" (FP) y "False Negatives" (FN) son patrones clasificados como de la clase incorrecta.

Tabla 2.1 Matriz de confusión para dos clases, P y N.

		Clase Esperada	
		P	N
Clase Predicha	P	True Positive (TP)	False Positive (FP)
	N	False Negative (FN)	True Negative (TN)

Existen varias formas de combinar los términos anteriores para evaluar la calidad de un clasificador. La más inmediata es utilizar la tasa de éxito en clasificación (o *accuracy*, Ecuación 2.1). Esta medida suele ser la que se usa para evaluar un sistema de clasificación completo.

$$\text{Tasa de Éxito} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

Cuando se evalúa una regla o un clasificador parcial, no suele hacerse en función de la tasa de éxito, que es una medida global, sino que suelen usarse expresiones que hacen referencia al conjunto de patrones que la regla clasifica. La notación típica asume reglas que clasifican patrones dentro de la clase P, de ahí que las expresiones hagan referencia a esta clase:

- Una posibilidad es utilizar la medida de Precisión (*precision*), dada por la Ecuación 2.2. Esta medida es interesante porque sólo tiene en cuenta los patrones clasificados de una clase, lo cual puede ser una medida adecuada de calidad por clase.

$$\text{Precisión} = \frac{TP}{TP + FP} \tag{2.2}$$

- Como medida complementaria a la precisión se puede usar la Sensibilidad (*sensitivity* o *recall*, Ecuación 2.3). Es una forma de evaluar la "cobertura" del clasificador; el nombre alude a que se calcula en qué medida la regla es suficientemente "sensible" para detectar los patrones de la clase P.

$$\text{Sensibilidad} = \frac{TP}{TP + FN} \tag{2.3}$$

- Una tercera medida que se usa habitualmente se denomina Especificidad (*specificity*, Ecuación 2.4). Se trata de un análogo a la precisión, pero tomando en cuenta los patrones de la clase N.

$$\text{Especificidad} = \frac{TN}{TN + FP} \tag{2.4}$$

El problema de medir la calidad de una regla que forma parte de un clasificador, en lugar de un clasificador completo, puede expresarse como encontrar un compromiso entre precisión y sensibilidad, ya que ambos objetivos suelen estar en conflicto. Por ejemplo, en un caso trivial en que hay tantas reglas como patrones, y cada regla asigna exactamente la clase esperada a su patrón, la precisión es siempre máxima (1.0); sin embargo, la sensibilidad de cada una de estas reglas es mínima. En el extremo contrario, se puede definir una regla que asigna una de las clases a todos los patrones conocidos; al hacer esto, la sensibilidad para los patrones de dicha clase es máxima, pero la precisión es mínima, puesto que clasifica el resto de los patrones como falsos positivos.

La forma convencional de abordar este problema es agregar ambas medidas en una única función, utilizando a veces pesos para ponderar cada uno de los dos factores que compiten. Esta aproximación es la que se utiliza en la mayoría de los trabajos en la literatura.

Recientemente, se está comenzando a utilizar técnicas multiobjetivo en clasificación. El principio consiste en no agregar los objetivos, sino buscar soluciones de compromiso. Para comparar la calidad de dos reglas, hay que comparar sus resultados en todos los objetivos que se persiguen de forma independiente, y establecer reglas (por ejemplo, el concepto de dominancia) que determinen un orden parcial entre las calidades de las reglas.

Un enfoque multiobjetivo que se menciona en la literatura de clasificación es el análisis mediante tablas ROC. Una tabla ROC es un diagrama en dos

dimensiones en el que se representa en el eje Y la tasa de verdaderos positivos (Ecuacion 2.5) y en el eje X la tasa de falsos negativos (Ecuacion 2.6). Un clasificador corresponde con un punto de dicho diagrama, siendo preferible la zona próxima al punto (0,1). Realmente constituyen representaciones de un espacio de dos objetivos en el que Tasa_{tp} debe ser maximizado y Tasa_{fp} minimizado.

$$\text{Tasa}_{tp} = \frac{TP}{TP + FN} \tag{2.5}$$

$$\text{Tasa}_{fp} = \frac{FP}{TN + FP} \tag{2.6}$$

Puede consultarse un estudio detallado de las características, ventajas y formas de uso de estas tablas en [Fawcett, 2004].

También está atrayendo un interés creciente la utilización de técnicas multiobjetivo para agregar las medidas de evaluación del clasificador anteriormente citadas. Es decir, se consideran objetivos del aprendizaje, de forma independiente, la tasa de éxito de clasificación, complejidad de la solución, coste computacional, y otras específicas del algoritmo que se use para la clasificación. En buena medida el éxito del campo procede del desarrollo reciente de algoritmos evolutivos de optimización multiobjetivo. Puede consultarse [Jin, 2006] y [Jin, 2007] para una revisión del estado del arte en este campo.

2.1.2. Clasificación mediante vecino más próximo

Los Clasificadores por Vecino más Próximo (Nearest Neighbor Classifiers) [Cover and Hart, 1967] se definen como aquellos que,a partir de un conjunto P de patrones etiquetados, que puede ser o no el conjunto completo de datos de entrenamiento disponible (E), asignan a cada patrón o instancia no etiquetada, $\mathbf{v}_i \in V$, la etiqueta del patrón o patrones de P más "próximos" a $\mathbf{v}_i$. Si se utiliza sólo el patrón más próximo de P, hablamos del clasificador 1-NN, y si se usan los K patrones más próximos, clasificador por K vecinos (K-NN).

La forma de determinar la proximidad entre patrones es un parámetro del clasificador, si bien se suele entender que se utiliza la distancia Euclídea sobre el espacio definido por los atributos de los patrones. Habitualmente

se considera que dicho espacio es continuo ($\Re^d$), siendo d el número de atributos; sin embargo esta definición es extensible a atributos discretos (etiquetados): en este caso, o bien se transforma el atributo en una dimensión más, o bien se extiende la definición de proximidad de manera que incluya alguna medida de la similitud entre los valores discretos.

El conjunto P puede coincidir, en el caso más simple, con el conjunto de patrones disponibles para entrenar el clasificador (E); en otros, el conjunto P es de cardinalidad más reducida, y sus elementos suelen denominarse "prototipos" o "vectores". En estos últimos casos el clasificador puede denominarse Clasificador mediante Prototipo más Próximo (Nearest Prototype Classifier).

Geométricamente, estos clasificadores realizan una partición del espacio definido por los atributos del problema en regiones, cada una de las cuales recibe una de las etiquetas de clase. Los límites entre dichas regiones se denominan "fronteras de decisión".

En [Devroye et al., 1996] puede consultarse el análisis estadístico del clasificador con K vecinos. Su fortaleza consiste en la prueba de convergencia del error de clasificación en condiciones límite: se demuestra que, cuando el número de patrones disponible converge a infinito, el algoritmo 1-NN converge a un error no peor que el doble del error bayesiano, que se considera el límite inferior alcanzable. Igualmente, cuando se utilizan K vecinos, se aproxima al error bayesiano, para cierto valor de K que crece en función del número de patrones disponible.

El análisis estadístico de estos clasificadores considera que permiten estimar la probabilidad de que un patrón $\mathbf{v}$ pertenezca a una clase C dada ($p(C|\mathbf{x})$). Se determina que dicha probabilidad es proporcional al número de patrones de la clase C entre los K vecinos más próximos. En consecuencia, la regla de decisión debe asignar al patrón $\mathbf{x}$ la clase más numerosa entre dichos K vecinos. Puede consultarse la relación de este clasificador con métodos de clasificación clásicos en [D.Michie and D.J.Spiegelhalter, 1994].

El concepto de proximidad o similitud requiere de la definición de una medida sobre el espacio de atributos de los patrones. Es habitual el uso de la distancia Euclídea, pero también hay otras opciones [Atkeson et al., 1997], como la distancia Euclídea ponderada, distancia de Minkowski, simple o ponderada, distancia de Manhattan, distancia de Camberra, etc. La selección de la medida de proximidad apropiada para un problema puede ser determinante para que el clasificador obtenga buenos resultados. En el Apartado 2.1.6

introduciremos algunos trabajos que extienden el uso del clasificador mediante vecino más próximo usando medidas de proximidad distintas de la distancia Euclídea.

Si no se especifica, se entiende que se considera sólo el patrón más próximo dentro de P para determinar la clase. Sin embargo, hay ocasiones en las que el uso de un solo vecino para la clasificación no es la mejor opción. El resultado de la utilización de un vecino con distancia Euclídea da lugar a fronteras de decisión poliédricas a igual distancia entre los patrones más próximos a ambos lados de la frontera. Se pueden generar fronteras de decisión más complejas, bien variando la medida de proximidad empleada, bien utilizando K vecinos. Existen varios métodos de K vecinos, entre los cuales podemos citar:

- K-NN puro (K-NN por votación, o Voting K-NN). Es el algoritmo NN que utiliza la clase más frecuente dentro de los K vecinos para determinar la clase del patrón que se quiere etiquetar. De este modo, las fronteras se suavizan, pues un patrón que se encuentre rodeado de patrones de otra clase tenderá a ser ignorado en la clasificación. Este mecanismo ayuda a hacer al clasificador menos sensible al ruido de clase.

- K-NN ponderado (Weighted kNN). El criterio de K vecinos por votación no tiene en cuenta si dichos K vecinos están o no próximos, en términos absolutos, al patrón que se quiere clasificar. Puede darse el caso de que un patrón influya en regiones remotas del espacio. Para evitar este efecto, este algoritmo modifica la la regla de K vecinos asignando a cada uno de dichos patrones un peso que depende de la distancia entre el prototipo considerado y el prototipo más próximo al patrón. De este modo, se corrige el efecto de los patrones más alejados.

- K* (KSTAR) [Cleary and Trigg, 1995], es un método en el que se aplica una medida de similitud distinta de la Euclídea, que en la práctica pondera la influencia de los vecinos en función de su proximidad al patrón que se quiere clasificar; el algoritmo permite ajustar mediante un parámetro (*blending parameter*) el número K de vecinos que se toman en cuenta al hacer la clasificación.

Los algoritmos basados en vecino más próximo tienen la característica de que no construyen un modelo que resuma la información de partida. El hecho de prescindir de este modelo tiene dos consecuencias negativas:

- Desde el punto de vista del rendimiento, la tarea de clasificar un patrón supone un proceso costoso, puesto que obliga a calcular la distancia a todos los patrones conocidos. A pesar de que existen procedimientos que evitan realizar la comparación completa de distancias [Arya and Mount, 1993], sigue siendo un factor de ineficiencia.

- Por otro lado, no se puede considerar que estos métodos realicen ninguna tarea de extracción de conocimiento. Es decir, no se genera ninguna abstracción que caracterice el problema o permita inferir su estructura o propiedades.

Especialmente el primer problema condujo al desarrollo de una serie de métodos que pretenden reducir el conjunto de patrones conocido, conservando únicamente aquellos elementos que son relevantes para la clasificación.

Existen numerosos métodos de este tipo, que podemos agrupar en los dos grupos siguientes, en la que seguimos la denominación que se utiliza en [Kuncheva and Bezdek, 1998]: métodos de selección de instancias o prototipos ("Instance Selection" o "Prototype Selection") y métodos de reemplazo de prototipos ("Prototype Replacement").

2.1.3. Algoritmos de Selección de Prototipos

La primera de estas técnicas lo constituirían los también llamados métodos de selección de prototipos o instancias (Prototype Selection o Instance Selection). Estos procesan el conjunto de datos relativamente extenso, ya etiquetados (conjunto de entrenamiento E), y retienen del mismo los datos que resultan más significativos (conjunto de prototipos P). Hay muchos estudios comparativos de los diferentes métodos, como [Brighton and Mellish, 2002] o [Wilson and Martinez, 2000].

El conjunto P resultante puede evaluarse en función de que conserve algunas de las propiedades siguientes, que se definen en función del conjunto de entrenamiento E:

- Training Set Consistency (TSC); es decir, la medida en que el conjunto de prototipos clasifica con exactitud el conjunto de entrenamiento.

- Decision-Boundary Consistency (DBC); es decir, en qué medida el conjunto de prototipos reproduce con exactitud la forma de la frontera de decisión que genera el conjunto de entrenamiento.

- Minimal Set, (MS); en qué medida el número de prototipos en el conjunto P es mínimo.

Las propiedades anteriores sólo tienen en cuenta la capacidad de representación del conjunto P respecto del conjunto original de datos disponibles E.

Debe considerarse sin embargo que los criterios anteriores no son los únicos que son de interés cuando el conjunto P se va a utilizar como clasificador. Los dos criterios siguientes pueden en algunos casos entrar en competencia con la capacidad de representación:

- Capacidad de Generalización; P será más adecuado en la medida en que sea capaz de proporcionar una clasificación correcta para un conjunto de datos no etiquetados V. Un conjunto de prototipos que representa con exactitud el conjunto de entrenamiento puede no presentar una buena capacidad de generalización al clasificar patrones que no estaban presentes en E. Este efecto se denomina "sobreentrenamiento".
- Tolerancia al ruido; medida de la forma en que la presencia de datos ruidosos en el conjunto E no genera errores en la clasificación del conjunto V. Igualmente, un conjunto de prototipos que representa con exactitud el conjunto E, representa igualmente todo error que hubiera en él. En ese caso, los errores se reproducen al evaluar los patrones de validación.

Los distintos métodos de selección de prototipos se pueden agrupar para su estudio en tres clases, en función del mecanismo por el que se llega a P: métodos clásicos, geométricos y evolutivos. A continuación describimos los más conocidos.

Métodos clásicos de selección de prototipos

Consisten en la selección de un subconjunto de patrones de entrenamiento en función de diversas heurísticas. Muchos comparten la característica de ser secuenciales (van evaluando los patrones iniciales uno a uno para retenerlos o descartarlos) y dependientes del orden de evaluación.

Según la forma en que se realiza el proceso de selección, éstos métodos se pueden agrupar en varias familias:

- Los que se basan en un muestreo aleatorio de los patrones conocidos como Rmhc [Skalak, 1994].
- Los que se basan en la retirada sucesiva y ordenada de patrones del conjunto de entrenamiento, como Drop1, Drop2 o Drop3. Puede consultarse una descripción de estos algoritmos en [Wilson and Martinez, 1997].
- Los que realizan esta operación en función de la propia regla de vecindad, entre ellos: Condensed Nearest Neighbor (CNN, [Hart, 1968]), Edited Nearest Neighbor (ENN, [Wilson and Martinez, 2000]), Reduced Nearest Neighbor (RNN, [Gates, 1972]) y los llamados Instance Based Learning Algorithms (IB1, IB2, IB3, IB4-5) [Aha et al., 1991] y [Aha, 1992].

Métodos geométricos de selección de prototipos

Existen métodos que buscan una aproximación más sistemática al problema, a través del uso de aproximaciones geométricas como las descritas en [Godfried T. Toussaint and Poulsen, 1984]. Estas técnicas consisten en generar grafos que establecen una relación de vecindad basada en la distancia entre los patrones. Una vez generado uno de estos grafos, el conjunto de prototipos se obtiene a partir del conjunto original eliminando todos aquellos patrones cuyos vecinos sobre el grafo son todos de su misma clase.

- El diagrama de Voronoi se define como la partición del plano que se genera al aplicar directamente la regla del vecino más próximo. Cada punto pertenece a la región de Voronoi del patrón más próximo, y las fronteras son los puntos que están a igual distancia de dos o más patrones.

  Se puede realizar selección de prototipos utilizando un diagrama de Voronoi generado por un conjunto de patrones E, de la siguiente forma:

  1. Se traza el diagrama de Voronoi para E. Sobre este diagrama, se denominan "vecinos" de un patrón a los patrones que comparten un segmento de frontera.
  2. Se recorren todos los patrones y se marcan aquellos que tienen vecinos de clases distintas a la propia.
  3. Se descartan todos los patrones no marcados, es decir, aquellos cuyos vecinos son todos de su misma clase.

4. El resultado P_v es el conjunto de prototipos.

El conjunto P_v conserva las propiedades TSC y BSC mencionadas anteriormente; sin embargo el conjunto final no es mínimo. Sobre la construcción del grafo de Voronoi existen numerosas referencias; las más detalladas son [Klein, 1989] [Aurenhammer, 1991]. El inconveniente de este método es que la construcción del diagrama de Voronoi es de un orden de complejidad elevado, y se hace impracticable en dimensiones altas.

- Existe otro método basado en un grafo similar, que se denomina grafo de Gabriel [Gabriel and Sokal, 1969]. Se construye uniendo entre sí los patrones de E que son "vecinos de Gabriel". El grafo de Gabriel no requiere la construcción del diagrama de Voronoi, y existen algoritmos que permiten construirlo de forma más eficiente que el diagrama de Voronoi en dimensiones elevadas. En [Matula and Sokal, 1980] se describen con más profundidad las características y algoritmos de construcción de este grafo.

  La relación de vecindad de Gabriel se determina construyendo una esfera cuyo diámetro es la recta que une los dos patrones; si dicha esfera no contiene ningún otro patrones de E, los dos patrones considerados son vecinos. Sobre este grafo de vecindad, el mismo algoritmo que se utiliza sobre el de Voronoi construye un conjunto P_g que es un subconjunto de P_v. A pesar de no conservar TSC ni BDC, los resultados en clasificación no son en la práctica muy diferentes, puesto que fundamentalmente se producen en las zonas "externas" al conjunto original de patrones E.

En [Godfried T. Toussaint and Poulsen, 1984] se propone un tercer método basado en otro grafo de vecindad (Relative Neighborhood), que, mediante el algoritmo anterior, produce como resultado un conjunto P subconjunto de P_g.

En líneas generales, los métodos geométricos tienen un alto coste computacional que los hace poco viables en problemas de clasificación en los que la dimensión del espacio en el que se definen los patrones, y el número de patrones de E, son elevados.

En la Figura 2.1 se muestra la forma en que los métodos anteriores (el de Voronoi y el de Gabriel) realizan la selección de prototipos sobre un mismo conjunto de patrones inicial (representado en la primera figura). En

la segunda figura se han trazado las regiones de Voronoi que corresponden a todos los patrones, y se han marcado con un círculo los patrones cuyos vecinos son de clase distinta. Se eliminarán el resto, como los que aparecen en la región sombreada, puesto que todos sus vecinos son de su misma clase, es decir, no forman parte de la frontera. En la tercera figura se muestra cómo quedaría el conjunto de prototipos final P. Se puede constatar que la frontera de decisión permanece invariada. En la cuarta figura, se muestra el grafo de vecindad de Gabriel, y los patrones que se seleccionarían mediante dicho grafo. En lugar de los nueve patrones que quedaban con el método de Voronoi, este otro método dejaría únicamente seis patrones (marcados con círculos), a costa de que no se conserve ya la misma frontera de decisión.

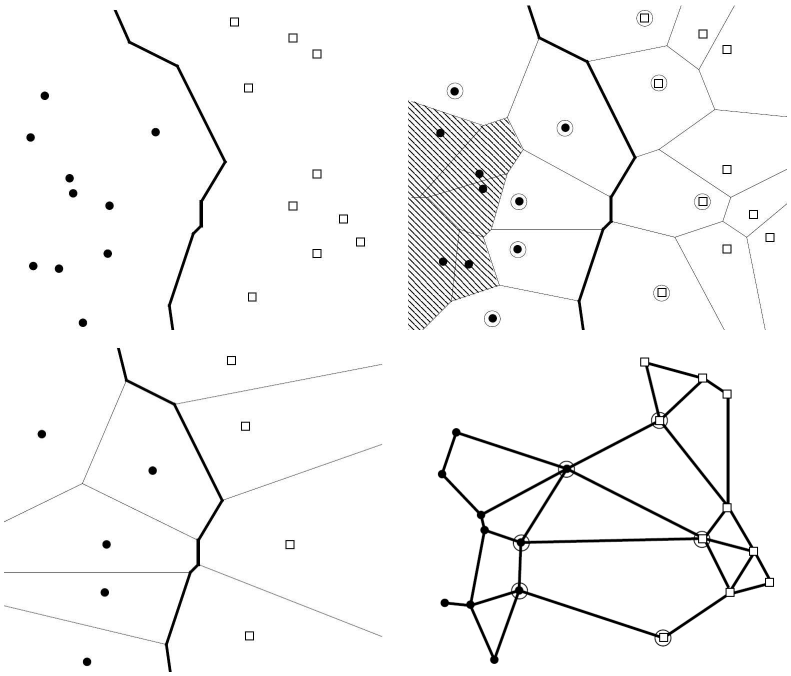

Figura 2.1 Selección de prototipos mediante métodos gráficos: arriba a la izquierda, conjunto completo T;arriba a la derecha, método de Voronoi: se eliminan los patrones que no están marcados con un círculo; abajo a la izquierda, conjunto P de prototipos y regiones resultado del método anterior, se conserva la frontera; abajo a la derecha, grafo de Gabriel y patrones (en círculo) que se conservan como prototipos. Nótese que en este último caso, se eliminan de P varios patrones con lo que la frontera de decisión resultante varía ligeramente (todas las figuras reproducidas de [Godfried T. Toussaint and Poulsen, 1984])

Métodos evolutivos de selección de prototipos

Existen también numerosos trabajos que utilizan algoritmos evolutivos para seleccionar el conjunto de instancias [Bezdek and Kuncheva, 2000].

Para realizar una aplicación de estos algoritmos, es vital determinar una función de evaluación (fitness) que evalúa las propiedades que se consideren relevantes para la aplicación. Por ejemplo, puede considerarse únicamente el error de clasificación, o puede incluirse como criterio el tamaño del conjunto P resultante. El problema se abordar habitualmente mediante algoritmos de optimización convencionales (es decir, con una sola función objetivo), combinando si es necesario los diversos objetivos en la función de fitness.

En ([Cano et al., 2003]) se analizan cuatro tipos de algoritmos evolutivos: el GA convencional (Generational Genetic Algorithm o GGA), Stationary Genetic Algorithm (SGA, actualiza sólo un par de individuos por generación), CHC Adaptive Search [Eshelman, 1990], y PBIL [Baluja, 1994]. Los resultados se comparan con las técnicas clásicas cuando se evalúa tanto el porcentaje de éxito como el porcentaje de reducción. Se utiliza una codificación binaria, en la que se marcan las instancias que deben formar parte del conjunto P. En consecuencia, a medida que crece $|E|$, es decir, el número de patrones del conjunto de entrenamiento, crece el espacio de búsqueda de los algoritmos evolutivos, que tienen dificultades de convergencia en algunos dominios. Sin embargo, los resultados son competitivos con los algoritmos clásicos.

Se han desarrollado también algoritmos evolutivos que usan operadores específicos para mejorar la capacidad de búsqueda; entre ellos se encuentra el propuesto en [Gil-Pita and Yao, 2008].

Existen también trabajos en los que se aplican técnicas de clasificación multiobjetivo que toman en cuenta los distintos criterios de evaluación del clasificador de forma independiente. Un ejemplo es [Chen et al., 2005], en el que se utiliza un algoritmo genético multiobjetivo para minimizar tres objetivos: el error de clasificación, el número de atributos seleccionados, y el número de prototipos usados por el clasificador.

2.1.4. Algoritmos de Reemplazo de Prototipos

Los métodos de reemplazo de prototipos realizan el análogo de un proceso de abstracción en el que, a partir del conjunto de datos original, se calcula un

conjunto de ejemplos de referencia que contiene las características esenciales de aquellos a los que representa. En este contexto, un prototipo no tiene por qué coincidir exactamente con ningún ejemplo de los disponibles en el conjunto original.

Estos métodos se denominan métodos de reemplazo, puesto que se sustituye la información inicial (conjunto de entrenamiento) por otra diferente. Algunos de estos métodos se analizan en [Kuncheva and Bezdek, 1998], con el fin de proporcionar un marco conceptual único para su estudio.

La forma de llegar al conjunto P de prototipos es variada y suele incluir al menos un parámetro, el número de prototipos ($|P|$).

Entre estos métodos encontramos aquellos que se denominan como métodos de Cuantización de Vectores (Vector Quantization). Entre ellos, los basados en el modelo de aprendizaje competitivo (Competitive Learning) como Learning Vector Quantization (LVQ, [Kohonen, 1995]) y otros derivados como Generalized LVQ (GLVQ, [Sato and Yamada, 1995]), o las Adaptive Quadratic Neural Networks (DR, [Lim et al., 1991]).

En líneas generales estos algoritmos operan modificando un conjunto de prototipos inicial actualizando la posición de los mismos al ir presentando uno a uno los patrones conocidos. Para mantener los prototipos dispersos, la regla de actualización da preferencia al prototipo "ganador" definido éste como el más similar (el más próximo) al patrón analizado. En LVQ, el algoritmo modifica la posición de un solo prototipo cada vez, mientras que en GLVQ o DR se pueden actualizar simultáneamente todos los prototipos. En estos dos algoritmos (GLVQ y DR) se especifica la función de coste que el algoritmo pretende optimizar; ello permite probar la convergencia de los algoritmos, a diferencia de lo que ocurre con los algoritmos LVQ básicos. Se puede consultar la discusión sobre este aspecto en [Sato and Yamada, 1995]. El grado de actualización suele depender de un parámetro adaptativo o tasa de aprendizaje, cuyo valor puede evolucionar en función del estado del algoritmo.

Existen diversas variantes de LVQ, que introducen conceptos adicionales con el fin de mejorar algunas de sus características. Entre ellas podemos citar:

- LVQ1 Algoritmo básico, se mueve un prototipo en cada iteración.
- OLVQ1 (Optimised LVQ1); como LVQ1, pero cada prototipo tiene su propia tasa de aprendizaje.

- LVQ2.1; se seleccionan los dos prototipos más cercanos y se actualiza su posición sólo si uno es de la clase del patrón y el otro no; además se requiere que la distancia entre ambos esté dentro de una ventana predefinida.

- LVQ3; similar a LVQ2.1 pero actualiza también los prototipos si son de la clase correcta.

- OLVQ3 (Optimised LVQ3); como LVQ3 pero cada prototipo tiene su propia tasa. de aprendizaje, como en OLVQ1.

Las distintas variantes introducen diferencias en la forma de posicionar los prototipos. Por ejemplo, LVQ2.1 intenta moverlos hacia la frontera de decisión; si bien de este modo alcanza mejor resultado que LVQ1, el algoritmo resultante es muy sensible al ruido. El autor propone utilizar varias pasadas del algoritmo , comenzando por LVQ1 y refinando el resultado mediante alguna de las versiones posteriores (Multi-pass LVQ).

También pueden plantearse algoritmos que ajustan el número de prototipos de forma adaptativa, durante la ejecución del algoritmo de reemplazo. Por ejemplo, OSLVQ [Cagnoni and Valli, 1994] incluye dos procesos complementarios: el GP (Growing Process) introduce prototipos en zonas donde se producen clasificaciones incorrectas; y el PP (Pruning Process) elimina prototipos que se evalúan como superfluos o redundantes. El algoritmo funciona como un LVQ estándar que se interrumpe en ciertos momentos para realizar esta "reconfiguración" de la red y proseguir. El algoritmo requiere sin embargo fijar valores para ciertos umbrales que condicionan los procesos de GP y PP.

Dentro de la familia de algoritmos de reemplazo de prototipos, procede mediante agregación de los patrones del conjunto de entrenamiento. Así opera el algoritmo propuesto por Chang (CA [Chang, 1974]). Se parte de un conjunto de prototipos igual al conjunto de entrenamiento, y se intenta agregar un par de prototipos de la misma clase, reemplazándolos por un punto intermedio, calculado como la media ponderada de ambos. Si el resultado de la agregación mantiene el error de clasificación, se da la fusión por realizada. Durante el proceso, cuando dos prototipos se fusionan, su peso se calcula como la suma de los pesos de sus padres, de modo que a medida que los prototipos se "concentran", sus posiciones se ven menos modificadas al agregarse a otros. Mediante la regla de agregación, el algoritmo CA tiende a producir prototipos situados cerca de los centroides de los agrupamientos

de patrones. En [Kuncheva and Bezdek, 1998] se propone una modificación sobre este algoritmo (MCA), que según los autores permite clasificar correctamente prototipos lejanos al centroide del agrupamiento. En este grupo podríamos también incluir algoritmos basados en métodos de clustering, como k-medias (original de 1956 pero publicado en [Lloyd, 1982]).

Desde el punto de vista de los algoritmos evolutivos, no existen tantas aplicaciones como en el campo de los métodos de selección de prototipos. Como ejemplo mencionamos el Evolutionary Nearest Prototype Classifier (ENPC, [Fernández and Isasi, 2004]). En dicho trabajo se propone un algoritmo que encuentra un conjunto de prototipos mediante la aplicación de operadores específicos a una población de posibles soluciones que comienza a partir de un solo prototipo. La ventaja de este método es que no requiere ningún parámetro, y en especial no necesita una estimación del número de prototipos $|P|$ adecuado. El enfoque en este caso no es el de un algoritmo genético clásico; el algoritmo se inspira en una metáfora biológica en la que los individuos de la población compiten por los recursos (patrones).

2.1.5. Algoritmos de clasificación que utilizan prototipos

Dentro de los algoritmos de clasificación o aprendizaje automático, hay algunos que utilizan, bien de forma implícita o explícita, un concepto similar al de prototipo, si bien no realizan la clasificación mediante la regla del vecino más próximo. Los que tienen un desarrollo más significativo son las Redes de Base Radial (RBFNN, [Powell, 1987]) y las Máquinas de Soporte Vectorial (Support Vector Machines, SVM [Boser et al., 1992], [Cortes and Vapnik, 1995]).

El campo de las SVM tiene un desarrollo muy importante tanto en aplicaciones de clasificación como de regresión. El principio de las SVM consiste en proyectar el espacio de los patrones en un espacio de mayor dimensión, donde se puede realizar una separación de los patrones de clases diferentes mediante superficies que se encuentran a máxima distancia de los patrones de las clases que separan. El concepto inicial de las SVM como clasificadores mediante hiperplanos es anterior al modelo actual en el que la separación se hace no lineal mediante funciones de kernel. Se puede consultar [Meyer et al., 2003] para una comparación con otros métodos de clasificación.

Se pueden interpretar las SVM como clasificadores basados en prototipos. Por ejemplo, en [Blachnik and Duch, 2008] se describe dicha interpretación y cómo se pueden usar técnicas de selección de prototipos para reducir

el número de vectores de soporte de la solución.

Igualmente se puede considerar que las RBFNN utilizan un mecanismo de prototipos subyacente cuando se utilizan en aplicaciones de clasificación. En una RBFNN cada neurona oculta tiene una posición (centro) y una desviación. Para determinar el nivel de salida de la neurona m cuando se le presenta el patrón k, la función de activación utiliza la distancia entre el centro de la neurona $\mathbf{c}_m$ y la posición (en el espacio de atributos) del patrón. El nivel de salida viene dado por la Ecuación 2.7, en la que δ_m es la desviación de la neurona m.

$$\phi_m(\mathbf{x}_k) = e^{-\frac{\|\mathbf{c}_m - \mathbf{x}_k\|^2}{2 \cdot \delta_m^2}} \tag{2.7}$$

Suele hacerse corresponder el número de neuronas ocultas con el número de clusters detectados en los patrones de entrenamiento, y la posición del centro con la posición del centroide del cluster, determinada con cualquier algoritmo de clustering, típicamente k-medias.

2.1.6. Clasificación con variación de la medida de proximidad

En los algoritmos que usan regla de clasificación por vecino más próximo, la medida de proximidad utilizada afecta de forma importante al resultado del clasificador. El caso más básico utiliza la distancia Euclídea, tal y como se define en la Ecuación 2.8. Se asume que los atributos constituyen un espacio F, en el que la función de proximidad se aplica a un par de vectores: el patrón $\mathbf{x}$ y un punto de "consulta" $\mathbf{q}$. Al usar prototipos, utilizaríamos cada uno de los prototipos como consulta. El índice f se usa para identificar la coordenada de cada vector correspondiente a un atributo.

$$d(\mathbf{x}, \mathbf{p}) = (\sum_{f \in F} |x_f - q_f|^2)^{\frac{1}{2}} \tag{2.8}$$

Expresión generalizada de la medida de proximidad

En [Wettschereck et al., 1997] se analiza la posibilidad de utilizar medidas diferentes de proximidad al clasificador K-NN de forma sistemática, utilizando para ello una expresión genérica de la función de proximidad que

incluye la Euclídea y otras medidas según la forma en que se escojan los parámetros (Ecuación 2.9).

$$d(\mathbf{x}, \mathbf{q}) = (\sum_{f \in F} w(f) \cdot \delta(x_f, q_f)^r)^{\frac{1}{r}} \quad (2.9)$$

En esta expresión aparece un exponente r , que toma el valor 2 en la distancia Euclídea; una función de ponderación $w(f)$ que devuelve un valor real o binario que se aplica para cada sumando de la expresión, que toma el valor 1 en las distancias no ponderadas; y una diferencia δ entre el valor del atributo f del patrón (x_f) y la consulta (q_f), que en la distancia Euclídea era el valor absoluto de la diferencia aritmética entre ambos valores.

La expresión generalizada permite tratar también el caso de valores de atributos discretos. Para ello se puede definir el valor de la función δ cuando opera sobre valores discretos de la siguiente manera: toma el valor 0 cuando los patrones tienen el mismo valor para el atributo, y 1 en caso contrario (medida City-Block).

Al considerar distintas posibilidades para la expresión anterior, aparecen variantes de de la familia de los clasificadores con la regla del vecino más próximo:

- Si la función w(f) toma la forma de una matriz unidad, el exponente $r = 2$ y la función δ es el valor absoluto de la diferencia entre las coordenadas de los dos vectores, la expresión 2.9 se reduce a la de la distancia Euclídea 2.8.
- Si se varía respecto al caso la función w(f) para que sea una matriz diagonal, la expresión resultante es equivalente a una distancia Euclídea con ponderación de atributos. Es decir, cada atributo tiene asociado un factor de ponderación (peso) por el que se multiplica antes de usarlo en el cálculo de la distancia.
- Si la función w(f) no corresponde a una matriz diagonal, estaremos utilizando una medida de proximidad ponderada. La expresión general puede entonces reescribirse como en la Ecuación 2.10. En dicha expresión, M es una matriz cuadrada de coeficientes reales, de dimensión $d \times d$, siendo d la dimensión del espacio F.

$$d_M(\mathbf{x}, \mathbf{q}) = \sqrt{(\mathbf{x} - \mathbf{q})^T \cdot M^T \cdot M \cdot (\mathbf{x} - \mathbf{q})} \quad (2.10)$$

- Si se varía el exponente r, manteniendo la función $\delta()$, se obtiene una distancia Ponderada de Minkowsky, con el exponente considerado.

- Si la función $w(f)$ es binaria, estamos en el caso de un método con selección de atributos; es decir, es un clasificador en el que sólo algunos de los mismos son relevantes.

- Si la función $w(f)$ no depende de $\mathbf{x}$ o $\mathbf{q}$, la medida de proximidad es *global* (idéntica para todos los puntos del espacio de atributos); también se pueden definir medidas *locales* , cuando dependen de la consulta (q) o del patrón (x) considerado. En este caso conviene sustituir la expresión de $w(f)$ por $w(f, \mathbf{x})$ o $w(f, \mathbf{q})$ para hacer explícita esta dependencia.

De hecho, la expresión 2.9 no es la única forma de expresar la similitud entre patrones del espacio de atributos, pudiendo tomar formas matemáticas más complejas. Igualmente existen otras medidas que se pueden aplicar para estimar la similitud entre atributos nominales, más complejas que la medida City-Block.

En [Atkeson et al., 1997] se realiza un extenso análisis de diversas medidas de proximidad no Euclídeas que el autor aplica con posterioridad a algoritmos de regresión.

Representación gráfica de las medidas de proximidad

La Figura 2.2) muestra cómo la introducción de la matriz M en la Ecuación 2.10 transforma la superficie de igual valor de dicha función, aplicando escalado y rotación. En el plano, la circunferencia pasa a ser una elipse orientada según los ejes, al usar una matriz M diagonal, o inclinada respecto a los mismos, al usar una matriz no diagonal.

La distancia de Minkowski (o norma L_r) se define mediante la Ecuación 2.11, en la que la función de ponderación es constante $w(f) = 1.0$, y que depende del parámetro r. También se pueden utilizar pesos en dicha distancia, dando lugar a la norma L_r ponderada.

$$L_r(\mathbf{x}, \mathbf{q}) = (\sum_i |(x_i - q_i)|^r)^{\frac{1}{r}} \qquad (2.11)$$

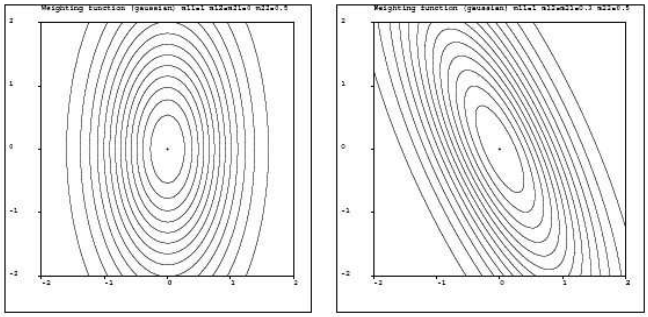

Figura 2.2 Trazado de las superficies de equidistancia, a la izquierda distancia Euclídea con matriz de ponderación diagonal, a la derecha distancia Euclídea Ponderada (figura reproducida de [Atkeson et al., 1997])

Sobre el plano, las superficies equivalentes a la circunferencia, en la distancia de Minkowsky, corresponden con curvas como las mostradas en la Figura 2.3.

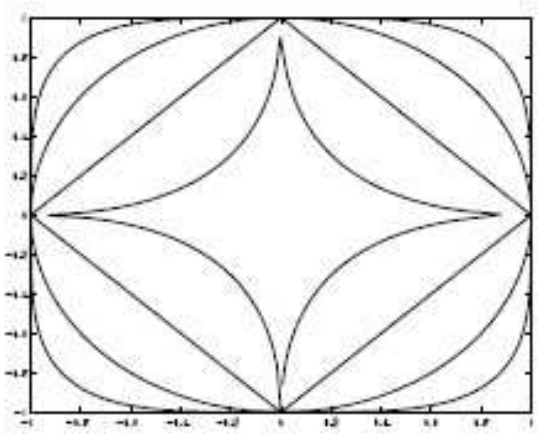

Figura 2.3 Trazado de las superficies de equidistancia, con la distancia de Minkowsky, para los exponentes $r = 0.5$,1,2 y 7 (del interior al exterior)(figura reproducida de [Duch and Grudzinski, 2001])

Con la introducción de distancias ponderadas, las fronteras de las regiones de Voronoi pueden ser curvas. En la Figura 2.4 se muestran ejemplos de

teselaciones de Voronoi para un mismo conjunto de prototipos de dos clases, al utilizar la distancia Euclídea (a), al usar una matriz M de proximidad diagonal pero no unidad ($[0.25, 0.0, 0.0, 0.8]$) para el patrón central (b) , y al variar las medidas de proximidad para todos los patrones (c y d). Se observa la curvatura que se introduce en (b) en los puntos próximos al patrón central. En c) y d) puede observarse cómo la medida de proximidad pierde su carácter intuitivo.

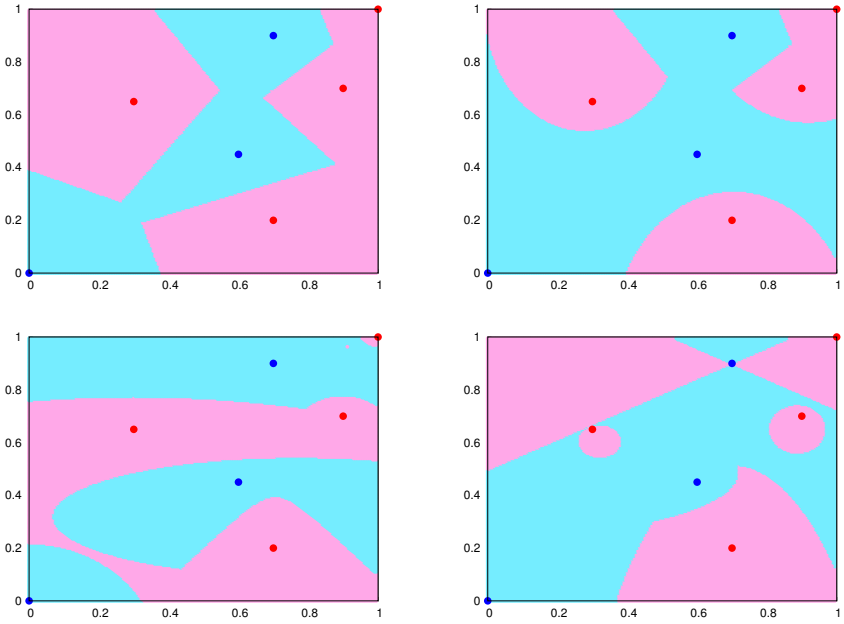

Figura 2.4 Trazado de las regiones de Voronoi correspondientes a siete prototipos (círculos) para distintos ejemplos de función de proximidad: a) distancia Euclídea ; b) Prototipo central con matriz [0.25,0.0,0.0,0.8]; c) y d) todos los patrones con matriz no unitaria

Utilización de medidas de proximidad no euclídeas en clasificación

Existen numerosos trabajos en los que se analiza el efecto de diferentes medidas de proximidad.

La variación de la medida de proximidad puede ser de utilidad en clasificadores basados en selección o reemplazo de prototipos, así como en otros clasificadores en los que se utiliza este concepto internamente aunque la regla

de clasificación no sea la del vecino más próximo.

En los algoritmos 1-NN o K-NN estándar la medida de proximidad puede considerarse un parámetro del algoritmo. En algunos algoritmos este enfoque está implícito, como en el caso de KSTAR [Cleary and Trigg, 1995], si bien en este algoritmo en la medida de distancia se pondera la contribución de cada prototipo a la clasificación de cada patrón (ver 2.1.2).

Existen trabajos en los que se aborda la determinación de la mejor medida de proximidad como un problema de optimización; dado el conjunto de datos, se busca la medida que mejor permite clasificarlo. En estos casos, se deben codificar los parámetros de una medida de proximidad como un individuo a optimizar. En caso de abordarse con un algoritmo evolutivo, la función de fitness consistirá en la aplicación de un algoritmo de clasificación que utilice dicha medida de proximidad.

Por ejemplo, en [Valls et al., 2005] y [Valls et al., 2007] se aborda este problema, con la intención de utilizar una medida de proximidad distinta de la Euclídea en la expresión que se utiliza en la función de activación de una Red de Base Radial (Ecuación 2.7).

Se compara la mejora obtenida por una medida con matriz diagonal y con una matriz simétrica no diagonal. Los coeficientes de la matriz $M^T \cdot M$ (Ecuación 2.10) se hacen evolucionar mediante un algoritmo genético convencional, de manera que optimicen el error de clasificación.

La restricción sobre el tipo de matriz que se evoluciona se impone en la codificación de la solución en el algoritmo genético, ya que sólo se utilizan los coeficientes necesarios de la matriz. La función de proximidad así obtenida se aplica de forma global a todas las neuronas de la red. Los autores determinan que la mejora es significativa tanto en el caso de matriz diagonal como en el de matriz simétrica, con mejores resultados de la segunda para algunos de los dominios considerados.

Otros trabajos se diferencian del anterior en el sentido de que la medida de proximidad se va calculando durante el entrenamiento del clasificador "subyacente". En estos trabajos la característica de mayor interés es el carácter adaptativo y dinámico de la función de proximidad calculada.

- En [Cost and Salzberg, 1993] se introduce una medida de distancia para atributos discretos en un clasificador por vecino más próximo convencional. Los valores numéricos para la distancia entre dichos atributos varían conforme el algoritmo evoluciona. Además, el algoritmo

calcula un vector de pesos para los atributos. Ello permite mejorar el rendimiento de los algoritmos básicos de vecino más próximo, así como el de clasificadores basados en otros principios, en algunos dominios.

- En [Gagnec and Parizeau, 2007] se aborda este problema introduciendo varias propuestas de interés. Se proponen inicialmente dos algoritmos independientes: un algoritmo multiobjetivo para selección de prototipos, y un segundo algoritmo basado en Programación Genética (GP, [Koza, 1992]) para calcular una medida de proximidad óptima dado un conjunto de prototipos.

  A continuación, se proponen dos enfoques coevolutivos, en el que ambos algoritmos se ejecutan en paralelo:

  - El primer algoritmo es de tipo cooperativo: la población determinada por cada uno de los algoritmos propuestos inicialmente se utiliza en el cálculo de fitness del otro. Una "especie" está formada por prototipos, y la otra por medidas de proximidad. En este caso ambas especies se benefician del crecimiento de la otra.
  - La segunda propuesta incluye una tercera "especie" que compite con las anteriores. En este caso, el objetivo de dicha especie consiste en seleccionar los patrones más difíciles de clasificar por el resto del algoritmo, estableciendo así una presión evolutiva adicional sobre las poblaciones iniciales.

- En [Fernandez and Isasi, 2008] se añade al cálculo adaptativo de la mejor proximidad el hecho de que ésta se calcula de forma **local**. El trabajo plantea mejorar el algoritmo de clasificación Evolutionary Nearest Prototype Classifier (ENPC [Fernández and Isasi, 2004]), utilizando una distancia en la que $w(f, \mathbf{q})$ es un vector de pesos para los atributos que varía para cada prototipo $\mathbf{q}$ generado por el clasificador.

  La forma de cálculo del vector de pesos de cada prototipo es la siguiente: se calcula en cada iteración de manera que las componentes del "vector de distorsión" de cada prototipo sean iguales. Este vector se crea como la diferencia media contribuida por cada patrón asignado a un prototipo dado. Una de las conclusiones del trabajo es que la inclusión de la ponderación local reduce drásticamente el número de prototipos generado por el algoritmo a la vez que reduce el error de clasificación.

2.2 Enjambre de Partículas (PSO)

2.2.1. Fundamentos del algoritmo PSO

El algoritmo de Optimización mediante Enjambre de Partículas (Particle Swarm Optimization o PSO, [Kennedy and Eberhart, 1995]) es un algoritmo bioinspirado, que se basa en una analogía con el comportamiento social de ciertas especies. Se reconocen dos influencias básicas de inspiración en PSO:

- El movimiento de bandadas de aves, en las que cada individuo se desplaza mediante reglas simples de ajuste de su velocidad en función de las observaciones que realiza sobre los individuos próximos en la bandada.

- Un modelo social, en el que cada partícula representa una creencia, y las influencias entre individuos, la aproximación de unos individuos a otros por difusión de dichas creencias.

En líneas generales, el algoritmo busca encontrar el óptimo global de una función (función de fitness) mediante el movimiento de un conjunto de "partículas" (enjambre) en un espacio definido por el número de parámetros de la función. Cada partícula es una posible solución, es decir, un conjunto de parámetros, que se "evalúa" calculando el valor de fitness que les correspondería. En dicho movimiento, las partículas utilizan información histórica (el resultado de su exploración pasada), así como información sobre sus vecinos (otras partículas del enjambre).

Por otro lado, para completar la especificación del algoritmo, hay que definir ciertos parámetros del algoritmo (criterio de parada, tipo de vecindario, valores de algunas constantes, forma de inicialización). El enjambre comienza disperso por el espacio de búsqueda, pero con el tiempo las partículas convergen hacia una zona reducida del espacio en la que intensifican la búsqueda de la solución.

Cada partícula se caracteriza por un vector de posición $\mathbf{x}_i$, un vector de velocidad $\mathbf{v}_i$, y una memoria que contiene el vector con la mejor posición de la partícula hasta el momento $\mathbf{p}_i$. El vector de posición codifica una solución al problema de optimización, de modo que cada coordenada corresponde al valor de uno de los parámetros de la función que se desea optimizar.

En cada iteración, la velocidad de cada partícula se actualiza utilizando la Ecuación 2.12. Dicha ecuación está escrita en forma escalar, de modo que, para cada vector considerado, se calcula la actualización en cada dimensión d de forma independiente. Puesto que los vectores varían en cada iteración, se utiliza el superíndice t para indicar la iteración a la que se refiere una variable. Los términos c_1 y c_2 son parámetros fijos del algoritmo, y ψ_1 y ψ_2 son factores aleatorios entre 0 y 1, que se calculan de forma independiente en cada iteración y para cada dimensión. Por último, $\mathbf{p}_g$ corresponde con la mejor posición del mejor vecino de la partícula. Existen varias posibilidades para la definición del vecindario de una partícula, que describimos en el Apartado 2.2.1.

$$v_{id}^{t+1} = v_{id}^{t} + c_1 \cdot \psi_1 \cdot (p_{id}^{t} - x_{id}^{t}) + c_2 \cdot \psi_2 \cdot (p_{gd}^{t} - x_{id}^{t}) \qquad (2.12)$$

Cuando la velocidad de todas las partículas se ha actualizado, esta se suma vectorialmente con la posición actual de la partícula para moverla a su posición para la siguiente iteración, según la Ecuación 2.13.

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \qquad (2.13)$$

En la Ecuación 2.12 se suelen distinguir tres términos, que se suelen denominar haciendo referencia a la analogía social del algoritmo:

- Inercia (v_{id}^{t}), que representa la tendencia de un individuo a seguir explorando en la misma dirección.
- Cognitivo o individual, $(c_1 \cdot \psi_1 \cdot (p_{id}^{t} - x_{id}^{t}))$, que representa la tendencia a permanecer próximo a posiciones (o creencias) que le dieron buen resultado en el pasado.
- Social, $(c_2 \cdot \psi_2 \cdot (p_{gd}^{t} - x_{id}^{t}))$, que representa la influencia que ejercen el resto de los individuo o la imitación del mejor individuo conocido.

Definición del vecindario

PSO modela la cooperación entre individuos mediante la definición de un vecindario para cada partícula, de modo que en cada iteración se selecciona un candidato de dicho vecindario para guiar la búsqueda de la partícula (término social de la Ecuación 2.12).

Según los autores, el algoritmo se diseñó inicialmente con un vecindario dinámico, dada la analogía con la bandada de aves. Sin embargo, pronto se determinó que, para los problemas de optimización considerados, era suficiente y más eficiente que el vecindario de cada partícula se definiera de forma **estática**: es decir, cada partícula tiene unos vecinos definidos a priori. En ello se aproxima a los algoritmos genéticos celulares, autómatas celulares, y varios modelos de computación neuronal.

En [Kennedy and Mendes, 2002], los autores definen y analizan distintos tipos de vecindario estático. La estructura de las relaciones entre las partículas se denomina "topología" del vecindario. En algunas topologías (por ejemplo, la topología en "anillo"), es preciso además definir la cardinalidad del conjunto de vecinos (los dos vecinos más próximos, los tres vecinos, etc.).

En líneas generales, las topologías más interconectadas distribuyen la información de la mejor solución del enjambre con mayor rapidez; ello produce una convergencia más rápida. Por otro lado, si esta velocidad es demasiado rápida, se incrementa la probabilidad de que no se explore completamente el espacio de búsqueda; en ese caso se puede producir la convergencia a un óptimo local en lugar de al óptimo global (convergencia prematura).

La topología Global Best (*gbest*) consiste en considerar que todas las partículas están interconectadas, de manera que la partícula que interviene en la Ecuación 2.17 es la partícula con mejor fitness del enjambre. Si dicha partícula está próxima, en la inicialización, a un óptimo local, será difícil que el enjambre escape al mismo. Por lo tanto, la topología *gbest* es la de máxima interconexión.

La investigación realizada en este campo confirma que la topología que *gbest* sólo resulta adecuada en casos de problemas de optimización unimodales, es decir, cuando no existe riesgo de convergencia prematura, ya que sólo existe un óptimo y es global.

En el extremo opuesto (baja interconexión), se define la topología Local Best ("*lbest*"), con una cardinalidad que suele ser reducida con respecto al número de partículas del enjambre. En muchos casos el número de vecinos se mantiene en 3 o 5, incluyendo a la partícula en cuestión (que sería vecina de sí misma). Habitualmente se imaginan las partículas unidas en un anillo, y simplemente cada partícula es vecina de las dos adyacentes (una a cada lado del anillo), o de las cuatro adyacentes (dos a cada lado).

En la Figura 2.5 se representan las dos topologías mencionadas. En el

trabajo referenciado ([Kennedy and Mendes, 2002]) se analizan de forma empírica otros posibles modelos, que sin embargo no han alcanzado una difusión importante en la comunidad de PSO.

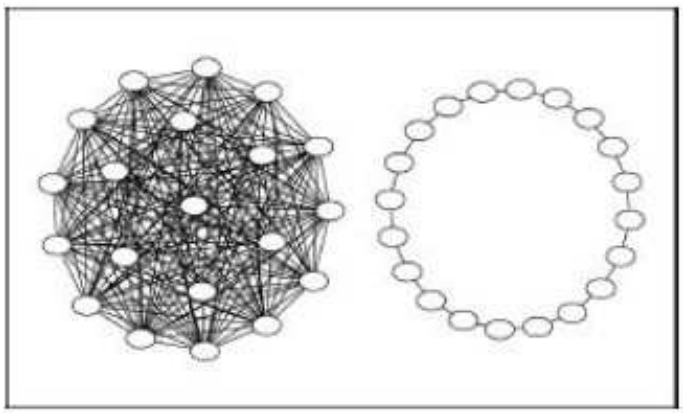

Figura 2.5 Topología de vecindario en PSO de tipo *gbest* (izquierda) y *lbest* (derecha), con las partículas representadas en un espacio de dos dimensiones (figura reproducida de [Kennedy and Mendes, 2002])

Existen también versiones de PSO con vecindarios **dinámicos**, en las que las relaciones entre las partículas evolucionan con el tiempo; si bien el uso de vecindarios dinámicos tiene un mayor coste computacional, puede ser interesante para ciertos tipos de problema.

Un trabajos en el que se estudia la posibilidad de un vecindario dinámico es [Suganthan, 1999]. Se propone un enjambre tal que el vecindario de cada partícula empieza siendo local (de hecho empieza restringido a la propia partícula) y crece gradualmente, incluyendo más vecinos, hasta que incluye todo el enjambre en las fases finales de la evolución del mismo. Es decir, evoluciona desde la topología *lbest* hasta la topología *gbest*. De este modo, el autor indica que, en fases iniciales de la evolución, en enjambre tiende a realizar búsqueda local simultánea en torno a las posiciones de las partículas, que estarán separadas en el espacio. En fases finales, el enjambre ha tenido ya oportunidad de localizar el óptimo global, y las partículas están próximas entre sí; en esta fase final se puede utilizar la topología *gbest* evitando el riesgo de convergencia prematura que se ha descrito con anterioridad.

Entre los vecindarios dinámicos, una modalidad que es de utilidad en ciertos problemas son los vecindarios **topológicos**; es decir, calculados en

función de la posición de las partículas sobre el espacio de búsqueda. Este concepto se utiliza típicamente en aplicaciones de optimización sobre funciones multimodales. Un ejemplo podría ser el propuesto en [Brits, 2002]. Al utilizar sólo las partículas próximas para dirigir la búsqueda, el enjambre puede converger de forma independiente a varios óptimos, si estos se encuentran separados de forma que las partículas que están en el entorno de óptimos distintos no ejercen influencia entre sí. En el Apartado 2.2.7 hacemos referencia a otros ejemplos de enjambres para optimización multimodal.

Por último, existen aplicaciones de PSO en las que el concepto de vecindario se define de forma más compleja. Un caso importante son los algoritmos PSO para optimización multiobjetivo. En estos, el propósito del enjambre consiste en optimizar varias funciones de fitness simultáneamente; por lo tanto, en general no es posible utilizar un criterio algebraico para determinar cuál es la mejor solución entre un conjunto de vecinos, sino que puede haber varias partículas que representen compromisos igualmente buenos entre los diversos objetivos (Conjunto de Pareto).

De manera más formal, en PSO multiobjetivo con n objetivos, la definición del vecindario se determina de forma dinámica considerando la posición de las partículas en el espacio $\Re^n$ formado por sus valores de fitness. Cada partícula tiene un valor de fitness para cada objetivo $(fit_1, fit_2 \cdots fit_n)$, y este conjunto puede interpretarse como un vector $\mathbf{f}$ en $\Re^n$.

Algunos ejemplos de aplicación de PSO a problemas multiobjetivo, y la determinación del vecindario que llevan implícita, son los siguientes:

- En [Hu and Eberhart, 2002] se propone un algoritmo para optimización multiobjetivo para dos funciones objetivo. El vecindario está definido por las partículas de fitness más próximo a la que se desea mover, para una de dichas funciones; y la selección del mejor vecino se hace atendiendo al valor de la segunda función de fitness. Es decir, una función de fitness determina la topología, y la otra el vecino considerado para la atracción.

- En Multiobjective PSO (MOPSO, [Coello et al., 2004]), el vecindario de una partícula se determina en función el conjunto de soluciones ya obtenidas por otras partículas en $\Re^n$. Al seleccionar un vecindario, se escoge un conjunto de partículas cuyos vectores $\mathbf{f}$ están en una región de $\Re^n$ específica: regiones del frente de Pareto en las que se han encontrado pocas soluciones. Entre dichas partículas "líderes" se

selecciona aleatoriamente la que atrae a la partícula que se estaba considerando.

2.2.2. Modificaciones al algoritmo original PSO

Durante la primera década de la utilización del algoritmo PSO, aparecieron numerosos trabajos que, partiendo del enjambre original, proponían modificaciones con el fin de mejorar sus características, normalmente de forma empírica. Asimismo aparecieron algunos trabajos que realizaron análisis teóricos de las ecuaciones del enjambre, con el fin de comprender mejor el mecanismo de su funcionamiento: la estabilidad, convergencia, estudio de trayectorias e influencia de los parámetros.

Algunos de los problemas de las primeras versiones de PSO se debían a su sensibilidad a la elección de los parámetros. Los valores escogidos no sólo condicionaban la posibilidad de encontrar el óptimo global de la función de fitness; se descubrió además el fenómeno de "explosión del enjambre". Éste consiste en que, con ciertos valores de los parámetros, las velocidades de las partículas ($\mathbf{v}_i$) tienden a crecer hasta valores elevados, dispersando el enjambre de forma que la distancia entre partículas era cada vez mayor.

Con el fin de estudiar la convergencia del algoritmo, y limitar la explosión del enjambre, se estudiaron varios mecanismos:

- Limitación de velocidad. Consiste simplemente en fijar un límite superior al valor absoluto de las componentes de velocidad. Es decir, se fijaba un parámetro V_{max} y se imponía la condición de limitación siguiente: $|\mathbf{v}_i^{t+1}| = minimo(|\mathbf{v}_i^{t+1}|, V_{max})$. Este mecanismo tiene el problema de que el parámetro V_{max} depende del tamaño del espacio de búsqueda (X_{max}), ya que en espacios de búsqueda grandes, velocidades pequeñas supondrían que el enjambre no recorrería todo el espacio o necesitaría un número de iteraciones elevado para hacerlo. En los trabajos iniciales se solía proponer el valor $V_{max} = X_{max}$.

- En [Shi and Eberhart, 1998] se introdujo un parámetro o factor de inercia, w (ver Ecuación 2.14). Se argumenta que el término de inercia de la ecuación es importante para que el enjambre pueda realizar búsquedas globales, más allá de las fronteras del espacio definido por las posiciones de las partículas iniciales. Ello se justifica por el hecho de que, con los parámetros inicialmente propuesto ($c_1 = c_2 = 2.0$), en

media el enjambre tiende a realizar una contracción hacia las mejores posiciones "interiores".

Los autores proponen incluir un parámetro explícito con el objetivo de permitir calibrar el equilibrio entre exploración y explotación, al añadir la posibilidad de variar el valor de dicho parámetro en función de la iteración o estado del algoritmo. En su análisis concluyen que, cuando el valor de w es pequeño, el enjambre tiene a comportarse como un algoritmo que realiza búsquedas local, y depende con más fuerza de la inicialización. Cuando w crece, pasa a realizar una búsqueda más global, pero encuentra el óptimo con más dificultad y mayor número de iteraciones. Mediante el estudio experimental realizado, se propone un valor de compromiso $0.9 < w < 1.2$.

$$v_{id}^{t+1} = w \cdot v_{id}^{t} + c_1 \cdot \psi_1 \cdot (p_{id}^{t} - x_{id}^{t}) + c_2 \cdot \psi_2 \cdot (p_{gd}^{t} - x_{id}^{t}) \qquad (2.14)$$

En el trabajo citado, se propone también el uso de un valor de w decreciente con el tiempo. Con posterioridad, esta propuesta se analizó de forma detallada en [Shi and Eberhart, 1999]. Aunque en este estudio el factor de inercia decreciente parece mejorar el comportamiento del enjambre, en [Zheng et al., 2003] se muestra un estudio en el que PSO tiene mejores resultados cuando se utiliza un valor de w que *crece* con el tiempo en lugar de decrecer. Por lo tanto, el comportamiento resulta ser dependiente de los problemas considerados.

- En [Clerc, 1999] se propone una alternativa a la inclusión del factor de inercia, que se estudia con más detalle en [Clerc and Kennedy, 2002]. En este trabajo se realizó un estudio teórico de la dinámica del enjambre y se observó que se podía garantizar la convergencia del algoritmo en ciertas condiciones mediante la inclusión de parámetros adicionales llamados "parámetros de constricción" (*constriction parameters*). Estos resultados se obtienen mediante el análisis de versiones simplificadas del enjambre, y específicamente versiones en las que se eliminan los factores aleatorios. Con posterioridad se confirman estas propiedades empíricamente usando la versión completa del enjambre. Los parámetros de constricción son una alternativa al mecanismo de inercia y de limitación de velocidad.

  En [Eberhart and Shi, 2000] se comparó la versión de PSO con constricción con la versión con inercia. La conclusión es que el método

con constricción es un caso particular del método con inercia, ya que se pueden calcular unos parámetros en función de otros. Sin embargo el método con constricción sigue requiriendo el uso de la limitación $V_{max} = X_{max}$.

En [Trelea, 2003] se realiza un análisis de la convergencia y un estudio de las trayectorias de las partículas en función de los valores de los parámetros. También se analizan aspectos como el tamaño de la población En el trabajo se confirman los valores que se recomendaban para los parámetros en estudios anteriores. Por otro lado, se afirma que las conclusiones obtenidas por el enjambre con factores aleatorios son similares a las obtenidas sin ellos, pero la inclusión de los mismos hace más lenta la convergencia e incrementa por lo tanto la exploración del espacio de búsqueda, previniendo así algunas situaciones de convergencia prematura.

2.2.3. Análisis teórico de PSO

PSO es un algoritmo de optimización basado en población, pero no un algoritmo evolutivo, de cuyo marco conceptual faltaría un proceso de selección entre las soluciones. Sin embargo, sí existe una relación entre las ecuaciones de PSO y algunos operadores de sobrecruzamiento en algoritmos evolutivos (Evolutionary Algorithms, EA [Baeck et al., 2000]) también usados para optimización numérica. El estudio de esta relación es interesante porque permite analizar las características de PSO utilizando terminología de dicho campo (algoritmos evolutivos).

Para visualizar esta relación, consideraríamos una meta-población compuesta por los individuos en sus posiciones actuales, así como la memoria de todos ellos (las mejores posiciones anteriores). En términos de EA, por cada individuo en la población "actual", se genera un "hijo" en la población de la iteración siguiente, producto de un operador de sobrecruzamiento que incluye tres padres, los tres vectores $\mathbf{x}_i$, $\mathbf{p}_i$, y $\mathbf{p}_g$. El nuevo individuo generado reemplaza al padre, manteniendo siempre el mejor de ambos en la "memoria" de la partícula como padre para siguientes generaciones.

El término de inercia (la influencia de $\mathbf{v_i^t}$ sobre la velocidad en el instante siguiente) corresponde con un factor de "momento" que se encuentra también en algunos algoritmos evolutivos, y que tiende a suavizar las oscilaciones en los movimientos de los individuos.

El concepto de vecindario, cuando éste es de tipo *lbest* y estático, existe

también en los llamados algoritmos genéticos celulares.

En cualquier metaheurística de búsqueda, evolutiva o no, cabe identificar los elementos que generan nuevas soluciones distintas de las encontradas hasta el momento (diversificación o exploración del espacio de soluciones), así como los que están encaminados a la mejora de las soluciones ya encontradas (explotación o intensificación). En PSO se puede entender que el término cognitivo o individual fomenta la explotación en torno a una buena solución potencial, y el término social fomenta la exploración. En este sentido, en ocasiones se ha mencionado la necesidad de incluir en PSO algún elemento que mantenga la diversidad de la población de forma explícita. Siguiendo la terminología de PSO, se ha introducido el término de *turbulencia* como en [Fieldsend and Singh, 2002] o, con terminología más habitual de algoritmos evolutivos, la inclusión de una operación de *mutación* como en [Esquivel and Coello, 2003] o [Zhang et al., 2005].

Existen varios trabajos en los que se analiza la trayectoria de las partículas en PSO. Para estos estudios suele considerarse un enjambre simplificado, compuesto por una sola partícula, y en los que el componente aleatorio se sustituye por un factor constante.

- El primer estudio se realizó en [Ozcan and Mohan, 1999], sobre las ecuaciones básicas de PSO, y se describe el movimiento de la partícula. En este trabajo se visualiza el movimiento de la partícula como un muestreo sobre una onda sinusoidal aleatoria.

- En [Clerc and Kennedy, 2002] (ya citado), el análisis de lugar a la introducción de términos de constricción y condiciones de convergencia.

- En [Trelea, 2003] se define el "enjambre determinista" con las Ecuaciones 2.15 y 2.16; el estudio considera por lo tanto la inclusión del factor de inercia (a). En la Figura 2.6 se reproducen algunas de las formas de trayectoria que se obtienen, en función de los valores de a,b, para ,$c = 1$ y $d = 1$. Se observa que, en todos los casos, la partícula se aproxima a la mejor posición, situada en la posición $y = 0$, pero puede hacerlo de forma suave, o con una oscilación amortiguada de diversas características, en función de los valores de dichos parámetros.

- En [van den Bergh and Engelbrecht, 2006] se realiza el análisis incluyendo el factor de inercia y se introducen los efectos de los términos aleatorios. Se determina que éstos introducen perturbaciones sobre el

comportamiento estudiado, que en general tienden a incluir cierta diversidad incluso en condiciones en las que el enjambre está próximo a la convergencia.

$$v_i^{t+1} = a \cdot v_i^t + b \cdot (p_{id}^t - x_{id}^t) \tag{2.15}$$

$$x_i^{t+1} = c \cdot x_i^t + d \cdot v_i^{t+1} \tag{2.16}$$

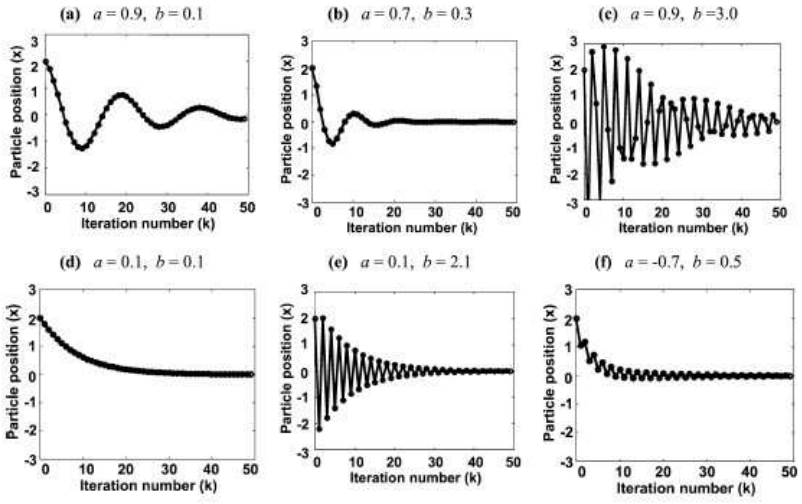

Figura 2.6 Representación de la trayectoria de una partícula aislada en un espacio unidimensional, para diferentes valores de los parámetros de PSO (figura reproducida de [Trelea, 2003])

Los análisis de trayectorias, como hemos visto, suelen ignorar la aleatoriedad del algoritmo PSO. Sin embargo, es posible abordar el estudio teórico de PSO considerando esta aleatoriedad, como en [Clerc, 2006] y [Jiang et al., 2007]. En ambos análisis se considera un enjambre en fase de "estancamiento' (*stagnation*), definida por que las partículas se comportan de forma independiente. Es decir, no se encuentran nuevos valores que modifiquen los puntos hacia los que las partículas son atraídas.

En [Poli and Broomhead, 2007] y [Poli, 2008b] se realiza un estudio de PSO utilizando Análisis de Momentos, en condiciones similares a los anteriores trabajos. Los autores caracterizan la distribución estadística del muestreo

que hace PSO de su espacio de búsqueda. Este tipo de estudio se realiza con el propósito de permitir análisis posteriores que permitirían caracterizar de forma sistemática, no empírica, el tipo de problemas cuyas características los hacen adecuados para su resolución mediante PSO.

2.2.4. Versión estándar de PSO

Con el fin de reunificar las propuestas tanto de ecuaciones de movimiento como de valores para los parámetros de PSO, en [Bratton and Kennedy, 2007] se pretende definir lo que debería considerarse una versión de referencia de PSO o "PSO estándar". Constituye un compendio de las mejores prácticas y propuestas para evaluar las futuras modificaciones del algoritmo que se propongan. Los autores del trabajo proponen usar dicho documento como base de comparación o evaluación para toda propuesta de modificación posterior y para las implementaciones de software de referencia.

Se ha adoptado una de las propuestas de [Clerc and Kennedy, 2002] como expresión estándar para PSO. Se introduce el parámetro χ o factor de constricción, de la forma que se indica en la Ecuación 2.17.

$$v_{id}^{t+1} = \chi \cdot (v_{id}^{t} + c_1 \cdot \psi_1 \cdot (p_{id}^{t} - x_{id}^{t}) + c_2 \cdot \psi_2 \cdot (p_{gd}^{t} - x_{id}^{t})) \tag{2.17}$$

El parámetro χ se deriva del resto de los parámetros mediante la Ecuación 2.18, en la que $\psi = c_1 + c_2$.

$$v_{id}^{t+1} = \frac{2}{2 - \psi - \sqrt{\psi^2 - 4\psi}} \tag{2.18}$$

Con esta expresión, con $\psi < 4$, el comportamiento del enjambre se considera que realiza una espiral hacia y alrededor de la mejor solución, sin garantía de convergencia, mientras que con $\psi > 4$ la convergencia sería rápida y garantizada. Por simplicidad, se escoge un valor $c_1 = c_2$ y $\psi = 4.1$, lo cual da lugar a los parámetros recomendados para el enjambre PSO estándar: $\chi = 0.72984$, $c_1 = c_2 = 2.05$.

Se analiza también el número de partículas recomendable, y se propone que éste varíe entre 20 y 50 en función del tiempo de ejecución o coste computacional deseado.

En cuanto a la topología del vecindario, se admite que, en la práctica, la topología *lbest* obtiene mejores resultados que *gbest* al evaluar el enjambre sobre problemas de cierta complejidad. La topología *gbest* se preferirá únicamente en caso de que el factor más importante de evaluación sea la velocidad de convergencia o el número de evaluaciones frente al riesgo de convergencia prematura.

En el enjambre PSO estándar se introduce un criterio de implementación de restricciones. Se recomienda que el enjambre no evalúe la función de fitness cuando la posición viola alguna restricción. De este modo, dicha posición nunca actualizará la mejor posición de la partícula. Con dicha modificación, los términos sociales de la ecuación de movimiento tenderán a hacer regresar a las partículas que violan las restricciones al espacio de soluciones válidas.

2.2.5. PSO Binario

En [Kennedy and Eberhart, 1997], los autores de PSO proponen una versión binaria del enjambre de partículas, con el objetivo de extender el campo de aplicación del algoritmo a problemas de optimización combinatoria. La modificación propuesta afecta a la codificación de las partículas, de modo que las coordenadas del vector de posición $\mathbf{x}$ pueden tomar únicamente los valores 0 o 1, y a la fórmula de actualización del mismo, que ya no es la suma vectorial con el vector de velocidad. En su lugar, se utiliza la Ecuación 2.19, en la que la posición se determina a partir de una distribución aleatoria de tipo sigmoidal, que depende de la velocidad. Para calcular la posición, primero se genera otro valor aleatorio, ψ_3 en el intervalo $[0, 1)$, que se compara con la función sigmoidal. Esta distribución está generada de forma que, a medida que la velocidad crece, la probabilidad de que la posición sea un 1 crece también.

$$x_{id}^{t+1} = \begin{cases} 1 & \text{si } \psi_3 < \frac{1}{1+e^{-v_{id}^{t+1}}} \\ 0 & \text{en caso contrario} \end{cases} \tag{2.19}$$

El enfoque anterior ha sido aplicado con éxito en algunos problemas de optimización, pero en líneas generales no ha sido un área en la que el Enjambre de Partículas se haya desarrollado demasiado, probablemente por falta de competitividad con algoritmos evolutivos de otras familias. Recientemente existen diversos trabajos de aplicación en Selección de Atri-

butos para problemas de clasificación (ver Apartado 2.2.7). Existen aplicaciones de este algoritmo estándar en la resolución del problema del viajante (Travelling Salesman Problem) [Zhi et al., 2004], Flow Shop Scheduling [Yuan and Zhao, 2007], y otros problemas clásicos.

La falta de consenso sobre la adecuación de la versión original de PSO binario para la resolución de problemas de optimización combinatoria ha dado lugar a numerosas propuestas alternativas; sin embargo, ninguna de ellas haya alcanzado por el momento un grado de consenso significativo y un número de aplicaciones suficiente como para reemplazar a la propuesta inicial:

- En [Shen et al., 2004] se propone el Modified Binary PSO (MBPSO). La modificación consiste en utilizar una estrategia diferente para la actualización de la posición de la partícula. Primero, se constriñe la velocidad al intervalo $[0, 1)$. Se divide dicho intervalo en tres segmentos, utilizando un nuevo parámetro umbral, α. Por último, según el segmento al que pertenezca la componente de velocidad analizada, la posición toma el valor anterior $\mathbf{x}^t$, el de la mejor posición pasada $\mathbf{p}_i^t$, o el del mejor vecino $\mathbf{p}_g^t$. Se puede interpretar como una forma de sobrecruzamiento aleatorio entre los tres vectores considerados.

- En [Clerc, 2005] se propone un enfoque genérico para diseño de enjambres PSO con codificación binaria, que no introduce la distribución de la posición en función de la velocidad, sino que define los operadores de la expresión de PSO estándar de forma que operen en espacios de coordenadas binarias. Así, un enjambre binario queda definido cuando se especifican los operadores siguientes:

  - "Modificar vector", que se aplica a los vectores $\mathbf{v}_i^t$, $\mathbf{p}_i^t - \mathbf{x}_i^t$ y $\mathbf{p}_i^t - \mathbf{x}_i^t$.
  - "Combinar tres vectores", que se aplica a los vectores modificados anteriores
  - "Combinar dos vectores", que se aplica al resultado anterior y la posición de la partícula $\mathbf{x}$.

  En líneas generales, la propuesta de Clerc consiste en catalogar las distintas familias de PSO binario pero no propone un algoritmo específico, dejando a trabajos posteriores la evaluación de una combinación de operadores que genere un algoritmo competitivo.

- Otro enfoque diferente es el propuesto en [Zhen et al., 2008]; en este trabajo se critica la falta de exploración local de MBPSO y se propone el PSO-based Probability Binary Optimizer, en el que se mantiene un vector $\mathbf{x}$ de números reales que se hacen corresponder con la probabilidad de que el valor de la solución, para dicha dimensión, sea 1. Al evaluar una partícula, dichos valores de probabilidad se utilizan para determinar el valor (0 o 1) que corresponde a la dimensión correspondiente.

2.2.6. PSO Discreto

En los trabajos originales de PSO no se prevé una versión discreta del algoritmo más allá de la versión binaria ya comentada. Sin embargo, aunque un algoritmo de optimización combinatoria se puede expresar mediante una codificación binaria, existe también la posibilidad de definir un enjambre de partículas cuyas posiciones y velocidades están definidas sobre un espacio de atributos discreto. Entendemos como tal espacio aquél en el que el valor de cada atributo tiene un valor entre un conjunto de valores disponibles, que pueden codificarse con un conjunto de índices o de etiquetas.

La aproximación más sencilla consiste utilizar velocidades continuas y posiciones discretas. Debe definirse algún método para transformar el espacio continuo de las velocidades sobre el espacio discreto de las posiciones. La ecuación de velocidad permanece como en PSO continuo, y el mapeo toma diversas formas según la aplicación considerada. Este tipo de enjambres discretos son análogos al enjambre PSO binario original (ver Apartado 2.2.5).

En [Tasgetiren et al., 2004] se propone una regla genérica, llamada Smallest Position Value (SPV) para transformar un espacio continuo en discreto. Sin embargo estas transformaciones tienen el problema de que secciones enteras del espacio de velocidades continuo representan la misma solución en el espacio de posiciones discreto.

Por el contrario, en [Clerc, 2000] se presenta la primera aproximación genérica a un enjambre PSO discreto, como una generalización del algoritmo PSO mediante la redefinición de los operadores necesarios para las ecuaciones del algoritmo. Las ecuaciones de velocidad y posición seguirían siendo las de PSO continuo, pero usarían los nuevos operadores. Para que las partículas se muevan en un espacio discreto es preciso definir los siguientes objetos y operadores:

- Posición ($\mathbf{x}$,$\mathbf{p}_i$,$\mathbf{p}_g$).
- Velocidad ($\mathbf{v}$).
- Diferencia entre dos posiciones, que debe generar una velocidad ($\mathbf{x} \ominus \mathbf{x}^{'} \rightarrow \mathbf{v}$).
- Producto escalar, produce una velocidad ($k \cdot \mathbf{v} \rightarrow \mathbf{v}^{'}$).
- Adición de velocidades, produce una velocidad ($\mathbf{v}_1 \oplus \mathbf{v}_2 \rightarrow \mathbf{v}_3$).
- Movimiento, combina una posición y una velocidad para producir otra posición ($\mathbf{x} \odot \mathbf{v} \rightarrow \mathbf{x}^{'}$).

El inconveniente de este enfoque es que es sólo una generalización que debe ser adaptada a cada aplicación concreta, y por lo tanto da lugar a familias diferentes de PSO discretos según la codificación y problema considerado. Como ejemplos, en [Clerc, 2000] se propone una versión para el problema del viajante (TSP), y en [Rameshkumar et al., 2005] una aplicación al problema de Flow Shop Scheduling.

El enfoque genérico anterior puede aplicarse para definir enjambres PSO discretos con posiciones y velocidades discretas. La nueva posición se calcula a partir de los elementos que intervienen en 2.12, pero interpretando las operaciones aritméticas como combinación sucesiva de sobrecruzamientos. El tipo de sobrecruzamiento depende del trabajo que se considere: puede ser aleatorio o por corte. La posición final es el resultado de combinar los tres elementos (posición anterior, mejor posición anterior y mejor posición del vecino como en el PSO original. Este enfoque se ha utilizado en [Pan et al., 2008] y [Zhang et al., 2008] para en problemas de Flow Shop Scheduling.

2.2.7. Aplicaciones del algoritmo PSO

El algoritmo PSO se ha empleado con éxito en numerosos problemas de optimización numérica. Se trata de una técnica de convergencia rápida e implementación simple, que obtiene resultados muy competitivos en muchos contextos, por lo que su difusión ha sido muy importante y los trabajos de aplicación numerosos en el campo de la optimización tradicional (con un sólo objetivo) (ver [Hu et al., 2004]); también en el campo de la optimización multiobjetivo [Hu and Eberhart, 2002] o [Zhang et al., 2003], optimización dinámica [Blackwell and Bentley, 2002] o [Blackwell, 2007].

Por otro lado, PSO tuvo en sus inicios un desarrollo importante como método de entrenamiento de redes de neuronas, en diversas aplicaciones [Eberhart and Shi, 1998], [Zhang and Shao, 2000].

Se ha realizado un análisis de publicaciones [Poli, 2008a] utilizando una base de datos de referencias, en el que pueden consultarse el número de trabajos en cada campo y su relación en forma de mapa conceptual.

Por su especial relevancia para nuestro trabajo, analizaremos en más detalle dos áreas de aplicación:

- Las aplicaciones que se han realizado en clasificación. Resultan más relevantes aquellas en las que el enjambre es el encargado de generar las reglas o prototipos con los que se realiza la clasificación; incluimos también referencias sobre utilización de PSO como optimizador para los parámetros de clasificadores basados en otros algoritmos.
- Las aplicaciones realizadas en Optimización Multimodal, ya que introducen el concepto de convergencia simultánea a varios óptimos. Este mecanismo es importante en algoritmo que proponemos más adelante (Capítulo 3, y por lo tanto resulta de interés analizar los mecanismos que se han propuesto para resolver este problema.

Clasificación mediante Enjambre de Partículas

Existen diversos trabajos en los que se utiliza PSO para generar reglas de clasificación, de los diversos tipos identificados en el Apartado 2.1.1: reglas basadas en predicados, reglas borrosas, o clasificadores por proximidad. A continuación comentamos los trabajos más relevantes realizados en dichos campos.

Uno de los primeros trabajos fue propuesto en [Sousa et al., 2004], donde se utiliza PSO para obtener reglas de clasificación definidas mediante intervalos para problemas con dos clases. En este trabajo, cada partícula codifica un intervalo para cada uno de los parámetros del problema de clasificación. Se reserva asimismo un intervalo de "indiferencia", que representa la situación en la que cualquier valor del atributo es válido. Cuando los valores de los parámetros de un patrón "caen" dentro de los intervalos definidos por la partícula, la regla le asigna la clase predominante en el conjunto de datos.

El proceso de clasificación es iterativo, es decir, se deja evolucionar el algoritmo hasta que se extrae una regla de inducción; los patrones del conjunto

de entrenamiento que son clasificados por esta regla se retiran, y el proceso vuelve a comenzar sobre el conjunto de datos de entrenamiento restante.

La función de fitness de una partícula es la calidad (Q_i^1) de la regla, definida según la Ecuación 2.20 (*Sensitividad* × *Especificidad*). Los resultados se comparan con un sistema de extracción de reglas basado en un algoritmo genético y con el algoritmo C4.5.

$$Q_i^1 = \frac{TP}{TP + FN} \times \frac{TN}{TN + FP} \tag{2.20}$$

Con una forma de codificación similar al trabajo anterior, existe un trabajo [Holden and Freitas, 2007] en el que se propone una versión híbrida de PSO y colonia de hormigas (Ant Colony Optimizacion, ACO). En dicho trabajo, PSO se usa para seleccionar reglas de inducción basadas en intervalos para los atributos numéricos del problema, y ACO para los atributos discretos. PSO también se usa, de forma simultánea, para determinar los parámetros numéricos del algoritmo ACO.

En este caso se experimentan diversas medidas de calidad de las reglas; resulta interesante que la función de fitness depende de la clase de la regla:

- Para la clase mayoritaria, los autores usan la función de calidad Q_i^1 (Ecuación 2.20) para predecir la clase mayoritaria
- Para las demás clases, se utiliza una medida diferente, basada en la *Sensitividad* × *Precisión* (Ecuación 2.21); luego modifican esta última para utilizar exclusivamente el valor de precisión corregida (Ecuación 2.22). En esta segunda versión, se usa el valor k (número de clases) para modificar el valor habitual de la medida de precisión ($TP/(TP+FP)$). De este modo se pretende, de forma simultánea, considerar soluciones precisas y "simples". Determinan que la mejor opción para los problemas que consideran es aquella que usa la precisión corregida (Ecuación 2.22).

$$Q_i^{3a} = \frac{TP}{TP + FN} \times \frac{TP}{TP + FP} \tag{2.21}$$

$$Q_i^{3b} = 1 + \frac{TP}{1 + k + TP + FP} \tag{2.22}$$

En trabajos más recientes como [Wang et al., 2006] y [Wang et al., 2007], se utiliza una aproximación similar. La función de fitness de nuevo está basada en la calidad Q_i^1 (Ecuación 2.20), aunque se añade un término que mide la comprensibilidad de la regla, es decir, se priman reglas que realizan el menor número de comparaciones posible.

También se puede utilizar PSO para optimizar un modelo de deducción borroso [Esmin, 2007]. Primero, se define una codificación que permita representar el clasificador *completo* en una partícula; en el caso de un sistema basado en reglas borrosas, esta representación incluye la definición de las funciones de pertenencia necesarias y las reglas del clasificador expresadas en función de los términos borrosos definidos para cada atributo. En este trabajo, el modelo está constituido por m entradas y una salida, la función de pertenencia de cada entrada incluye tres términos borrosos triangulares, y cada partícula codifica un conjunto completo de reglas (todas las combinaciones posibles de los términos borrosos). En este caso, la posición de la partícula contiene los valores de:

1. Los límites de los términos triangulares en cada función de pertenencia (entradas y salida)
2. La salida que corresponde a cada posible regla

Para evaluar el clasificador se utiliza la función de error cuadrático medio entre el valor predicho y el esperado.

Resulta de interés la propuesta de [de Almeida and Pozo, 2007]. En este caso, el problema planteado consiste en generar reglas de inducción para problemas con atributos discretos. La forma de realizarlo consiste en utilizar un algoritmo PSO multiobjetivo, de forma que, para cada regla, representada por una partícula, se evalúa simultáneamente un par de criterios. Se experimenta con distinto par de objetivos: precisión sobre ejemplos positivos y negativos en un caso, y sensitividad y especificidad en otro. Las partículas evolucionan de forma que se obtiene un conjunto de reglas que representan las mejores soluciones de compromiso entre los dos objetivos considerados. Los resultados son difíciles de evaluar, pues los autores transforman los conjuntos de datos que usan para reducirlos a dos clases. Las aportaciones fundamentales del trabajo son el uso de un enfoque Michigan (una regla por partícula) y el criterio de evaluación multiobjetivo de las reglas. También en [Ishida et al., 2008] se utiliza un enfoque multiobjetivo en clasificación mediante reglas, usando para evolucionar las soluciones el algoritmo MOPSO([Coello et al., 2004]).

Existen otros trabajos que no usan reglas de inducción, sino reglas de proximidad. Como se trata en el Apartado 2.1.2, los clasificadores por proximidad obtienen buenos resultados en numerosos dominios, especialmente en casos en los que el conocimiento sobre la estructura de los datos es limitado. Estos clasificadores están más próximos a la estrategia de clasificación que utilizaremos el algoritmo PSC que proponemos más adelante, ya que todos ellos son aplicaciones de clasificación por vecino más próximo con reemplazo de prototipos.

- En [Falco et al., 2006] se utiliza PSO para localizar un centroide por cada clase de un problema de clasificación. En este caso, cada partícula contiene tantos centroides como clases. Como medida de fitness, se usa la Ecuación 2.23. En ésta se divide el conjunto de entrenamiento E en tantos subconjuntos E_i atendiendo a la clase de los patrones; luego, para cada clase i, se suman las distancias desde cada patrón de entrenamiento de dicha clase ($\mathbf{p}_k$) hasta el centroide que le corresponde ($\mathbf{centroide}_i$). El total se divide por el número de patrones del conjunto de entrenamiento que son de dicha clase ($|E_i|$). Esta medida de calidad es interesante porque no calcula directamente la calidad de la clasificación. El clasificador resultante está limitado a un solo centroide por clase.

- En [Iswandy and Koenig, 2008] se utiliza PSO para refinar un conjunto de prototipos para su uso como clasificador por vecino más próximo. En este caso el conjunto se genera mediante algoritmos diferentes, y PSO sólo se utiliza a modo de optimización sobre la solución de los algoritmos anteriores. Cada partícula codifica el conjunto completo de prototipos.

- En [Nanni and Lumini, 2009] se codifica un conjunto fijo de prototipos con una aproximación convencional, similar a la que utilizamos en este trabajo como medida de comparación (Apartado 3.2). Los autores determinan que el algoritmo PSO por sí solo no resulta competitivo, pero en algunos dominios puede mejorarse el resultado entrenando secuencialmente un conjunto de clasificadores independientes y empleando luego este conjunto mediante un sistema de votación.

$$Q_i^2 = \frac{1}{|E_i|} \times \sum_{\mathbf{p}_k \in E_i} d(\mathbf{centroide}_i, \mathbf{p}_k) \qquad (2.23)$$

Existe otro trabajo que resulta interesante para nuestra aproximación, dado que la codificación del problema resulta muy similar. Se trata de la propuesta de [O'Neill and Brabazon, 2008], que combina conceptos extraídos de PSO con Self-Organizing Maps (SOM, [Kohonen, 1995]). El resultado se denomina Self Organizing Swarm (SOS). Si bien la aplicación resultante es una aplicación de agrupamiento (*clustering*), citamos este trabajo porque la codificación de un prototipo (neurona) por cada partícula, y su movimiento siguiendo ecuaciones de PSO, se aproxima al trabajo realizado por nosotros.

En SOS, las partículas se organizan topológicamente en una matriz rectangular que determina la relación de vecindad entre ellas. El algoritmo procede de forma incremental, como en SOM. Así, se considera secuencialmente el conjunto de patrones de entrenamiento; para cada patrón, se determina una partícula "ganadora" como la partícula más próxima al mismo; finalmente, la partícula ganadora actualiza su posición utilizando la posición del patrón como guía y una versión de las ecuaciones de PSO (Ecuación 2.14). Además, cuando se desplaza una partícula, se influye también en la posición de las partículas vecinas.

En conjunto, se trata de un SOM en el que las reglas de posicionamiento de las neuronas de la capa de mapeo utilizan los conceptos de inercia y memoria de PSO. Sin embargo, a diferencia de lo que ocurre en PSO, la interacción se produce entre partículas y patrones, y se mueve sólo la partícula que resulta ser más próxima al patrón que se considera en cada momento.

Existen otros trabajos de aplicación de PSO al problema de clustering, que aportan ideas que son de interés para el desarrollo de clasificadores. En [Omran et al., 2006] se utiliza el enjambre PSO binario en combinación con un algoritmo de clustering para determinar el número óptimo de clusters.

Enjambres de partículas para optimización de clasificadores

Existen otros trabajos en los que PSO se utiliza en combinación con otros clasificadores, de modo que contribuye más o menos estrechamente a la mejora de los mismos. En este caso PSO no se usa para encontrar las reglas del clasificador. Describimos a continuación algunos ejemplos.

Ya en los primeros trabajos con PSO se propone éste cómo técnica de entrenamiento de redes de neuronas [Kennedy et al., 2001]. En este sentido cabe proponer una red de neuronas para clasificación que usa PSO para determinar bien los pesos de las conexiones, bien la arquitectura de la red,

bien la combinación de ambas. En [Settles and Rylander, 2002] se utiliza el enjambre para optimizar únicamente los pesos de una red que se utiliza en clasificación. Se propone una versión multienjambre en la que varios enjambres cooperan en la resolución, para mejorar los resultados iniciales.

Una forma de optimizar el resultado de un clasificador consiste en procesar previamente los datos disponibles, seleccionando por ejemplo los atributos relevantes para la tarea de clasificación (Feature Selection). Un preproceso de este tipo es especialmente útil cuando el clasificador que se va a utilizar es un clasificador por vecino más próximo, que es sensible por tanto a la dimensionalidad del espacio de atributos. Se suele codificar la solución al problema como una cadena binaria, que contiene un valor 1 en la posición de los atributos que se consideran relevante.

Existen varios trabajos que realizan esta selección de atributos mediante un PSO binario. Es una aplicación normal del algoritmo descrito en el Apartado 2.2.5, pero el fitness de la partícula se calcula mediante la aplicación de otro clasificador "subyacente" (no basado en PSO), que considera únicamente los atributos seleccionados por la partícula. Algunos de los trabajos en este campo son los siguientes:

- En [Liu et al., 2006], PSO se usa para determinar el conjunto de atributos y los centros y número de neuronas ocultas de una Red de Base Radial (RBFNN) utilizada en clasificación.
- En [Correa et al., 2006] se propone una versión específica de PSO discreto para selección de atributos, en la que las partículas codifican combinaciones de índices de atributos sin repetición; se utiliza un clasificador bayesiano para determinar la clasificación óptima.
- En [Marinakis et al., 2008], se utiliza la versión binaria de PSO tal y como se introdujo en [Kennedy and Eberhart, 1997] para selección de atributos, combinado con un clasificador K-NN para clasificación.
- En [Chuang et al., 2008] se propone una nueva versión binaria de PSO denominada Improved Binary PSO, (IBPSO), en la que la mejor posición del enjambre se reinicializa si no varía en un determinado número de iteraciones. Se aplica este algoritmo para selección de atributos y clasificación posterior mediante la regla 1-NN.
- En [Melgani and Bazi, 2008] o [Huang and Dun, 2008], se usa un PSO convencional para optimizar simultáneamente el conjunto de atributos y los parámetros de una máquina de soporte vectorial (SVM).

Enjambres de Partículas para optimización multimodal

Las funciones multimodales se definen como aquellas que tienen más de un óptimo local, pudiendo además tener uno o varios óptimos globales.

Si sólo hay un óptimo global, el problema consiste únicamente en evitar la convergencia prematura a un óptimo local; ello se puede lograr mediante la introducción de mecanismos de diversificación.

- Una de las primeras aplicaciones de PSO a este tipo de problemas multimodales se encuentra en [Kennedy and Spears, 1998]. Para los algoritmos genéticos tradicionales (GA), este tipo de problemas puede ser difícil debido a que las operaciones de sobrecruzamiento pueden resultar destructivas cuando se producen entre individuos próximos a óptimos distintos. En este trabajo, se compara PSO con un GA convencional (con mutación, sobrecruzamiento y selección); y también con dos versiones simplificadas, una que no tiene mutación y otra que no incorpora sobrecruzamiento, por si éste fuera destructivo. En todos los casos, el PSO resulta ser competitivo con los GA.

- En [Esquivel and Coello, 2003] se usa un algoritmo PSO híbrido que incluye un operador de mutación propuesto por Michalewicz en 1996; esta modificación mejora el rendimiento de PSO comparado con la versión clásica. Tanto la versión clásica como la mejorada resultan competitivas al comparar con otras técnicas.

Un segundo caso, que resulta de más interés, es aquel en que hay más de un óptimo global. Este caso requiere incorporar mecanismos específicos para conseguir que un algoritmo encuentre dichos óptimos globales, ya que en un problema de estas características, el enjambre PSO estándar se comportará de una de las siguientes formas [Parsopoulos and Vrahatis, 2001, Parsopoulos and Vrahatis, 2002]:

- Encontrará un sólo óptimo

- Las partículas se moverán entre varios óptimos sin establecerse en ninguno de ellos.

Existen dos procedimientos para eliminar estos inconvenientes:

- El primero consiste en transformar la función de fitness cuando se localiza un óptimo (posiblemente local). Ello se hace mediante una transformación de "aplanado" que hace desaparecer el óptimo, de modo que el enjambre pueda continuar localizando el resto. La solución es la secuencia de óptimos encontrados. Las funciones de transformación pueden ser de diversos tipos ("function deflation", "function strechting)" [Parsopoulos and Vrahatis, 2001].

- El segundo resulta más relevante para nuestro estudio, pues consiste en intentar encontrar todos los óptimos simultáneamente; para ello es preciso realizar una partición del enjambre, de modo que algunas de las partículas se centren en unas áreas del espacio de búsqueda, y otras en áreas distintas. Las técnicas utilizadas suelen conocerse con el nombre de "técnicas de nichos" (*niching*). Existen algunos ejemplos de uso de estas técnicas en PSO:

  - En algunos trabajos se ha desarrollado el llamado Niching PSO [Brits et al., 2002], [Nickabadi et al., 2008]. En este, se divide el enjambre en "subenjambres" , que siguen a partículas líder diferentes; de este modo, buscan posiciones óptimas en torno a punto distintos. Igual que se produce la división del enjambre, existe la posibilidad de fusión de subenjambres cuando se detecta que están realizando búsquedas en torno al mismo objetivo. Los procesos de división y fusión se realizan de forma dinámica, en función de medidas que se realizan durante la iteración del algoritmo.
  - En [Passaro and Starita, 2008] se aplican técnicas de clustering para determinar explícitamente los vecindarios; ello se basa en la premisa de que, antes de que las partículas del PSO converjan por la influencia del óptimo global, se agruparán en torno a regiones donde es probable que existan óptimos locales. Para hallarlos, se combina un PSO estándar con un vecindario dinámico determinado por un algoritmo estándar de clustering.
  - En [Li, 2004] aparece el concepto de división en "especies". Cada especie utiliza un líder diferente, y cada líder intenta dirigir la búsqueda en torno a un punto del espacio distinto. En cada iteración del algoritmo, se recalcula la división en especies, así como la selección de la partícula líder para cada una de las especies.

De forma más genérica, observamos por lo tanto que, para que un en-

jambre pueda localizar varios objetivos simultáneamente, debe utilizarse una relación de vecindario entre partículas tal que el enjambre aparezca dividido, comportándose como si hubiera varios "subenjambres".

Por otra parte, a falta de información previa sobre la localización de los óptimos de la función de fitness, dicha partición debe ajustarse de forma dinámica, bien en función de alguna condición impuesta sobre las partículas que forman una clase, bien simplemente recalculándola iteración a iteración. Ello obviamente añade un coste computacional al algoritmo PSO estándar.

En tercer lugar, la forma de repartir el espacio obedece a criterios de localidad; especialmente al concebir la idea de "subenjambres", se usa una medida del "tamaño" de cada subenjambre para determinar si se debe producir división o fusión.

Estos tres conceptos son de aplicación al algoritmo que proponemos, puesto que comparte el objetivo común de encontrar buenas posiciones simultáneamente en regiones dispersas del espacio de búsqueda.

3

Clasificador mediante Enjambre de Prototipos

En este capítulo abordamos la justificación y definición del Clasificador mediante Enjambre de Prototipos (Prototype Swarm Classifier, o PSC).

Para ello, partiremos del algoritmo PSO ya conocido, y describiremos un posible método de resolución de problemas de reemplazo de prototipos para clasificación. Consideramos que esta forma de resolución directa, basada en un "enfoque de Pittsburgh", tiene serios inconvenientes incluso en problemas de clasificación sencillos.

A partir de las desventajas de dicha aproximación, describiremos el algoritmo PSC como el resultado de aplicar a PSO un nuevo enfoque en la codificación, que requiere una completa redefinición de algunos de los conceptos de PSO.

En la descripción de estos algoritmos incluimos los aspectos de codificación, ecuaciones y secuencia detallada de los mismos, en pseudocódigo.

Añadimos al final de este capítulo la descripción detallada de una serie de mejoras y variantes del algoritmo, que se proponen con el fin de mejorar sus características y que son objeto de experimentación en capítulos posteriores.

3.1 Principios Generales

El problema de reemplazo de prototipos consiste, como se ha descrito con anterioridad, en encontrar la posición óptima de un conjunto de prototipos, a partir de un conjunto de patrones de entrenamiento conocidos, de modo que se maximice la tasa de éxito de clasificación cuando se utiliza la regla

del vecino más próximo para asignar clases a patrones desconocidos.

Cabe pensar en una aplicación directa del algoritmo PSO estándar a la clasificación mediante prototipo más próximo; bastaría codificar la posición de los prototipos en la partícula secuencialmente, incluir la clase a la que corresponde el prototipo, y transformar el problema en un problema de optimización de la función "error de clasificación".

En ese caso, cada partícula representaría una solución completa, es decir, un conjunto de prototipos. La dimensión de la partícula sería igual al número máximo de prototipos multiplicado por el número de atributos en los patrones. Esta aproximación, llamada también enfoque de Pittsburgh, se describe en detalle en el Apartado 3.2.

Sin embargo, dicha aproximación tiene ciertos inconvenientes:

- En primer lugar, la dimensión de la partícula. Si se desea codificar un conjunto de prototipos (y no sólo uno) en una partícula, el espacio de búsqueda crece de forma proporcional al número de prototipos que se desea utilizar en el clasificador.
- En segundo lugar, la determinación del número de prototipos que se codifican en una partícula debe hacerse a priori, puesto que determina la dimensión de la misma. No es posible tampoco alterar el número de prototipos durante la ejecución del algoritmo, de modo que el algoritmo no puede determinar este parámetro de forma adaptativa.

Este enfoque se probará mediante una experimentación preliminar que sirva como referencia para el desarrollo de un tipo de enjambre más adecuado para la resolución del problema.

El objetivo es por lo tanto en encontrar un enfoque diferente, que permita utilizar conceptos de la metaheurística PSO pero ofreciendo características competitivas con otros algoritmos. Este nuevo enfoque lo denominamos Clasificación mediante Enjambre de Prototipos (CEP o en su traducción inglesa, PSC). Esta alternativa se describe en profundidad en el Apartado 3.3.

El PSC se fundamenta en utilizar una representación en la que coincide el espacio de búsqueda del algoritmo con el espacio de parámetros de la soluciones. Ello significa que cada solución del algoritmo corresponde exactamente con un prototipo de la solución.

Esta aproximación a la codificación de las soluciones es la que se conoce como "Enfoque de Michigan" [Holland, 1976] en el contexto de los sistemas

de clasificación basados en algoritmos genéticos. Esta forma de representación da lugar a la familia de técnicas que se suele denominar Learning Classifier Systems (LCS), de los cuales el más conocido es XCS [Wilson, 1995]. El principio de todos ellos es la evolución de microclasificadores que contribuyen de forma cooperativa al resultado global del clasificador. La familia LCS de clasificadores incluye numerosas variantes, basadas en general en aprendizaje por refuerzo. Se puede consultar una revisión del estado del arte en este campo en [Bernadó-Mansilla and Garrell-Guiu, 2003] y [Sigaud and Wilson, 2007].

En el caso de PSC, las ventajas fundamentales del enfoque de Michigan, respecto del enfoque descrito con anterioridad son:

- La dimensión de la partícula es igual al número de atributos de los patrones, y no depende del número de prototipos de la solución.
- Se puede variar el número de prototipos de la solución simplemente variando la población del enjambre: añadir o eliminar partículas.

Por otro lado, aparecen ciertas desventajas comunes a cualquier algoritmo de enfoque Michigan que hay que tener en cuenta [Wilson, 1995]:

- La función que determina la adecuación de un individuo depende de la composición del resto de la población; esto diferencia de forma fundamental el proceso de optimización, ya que, desde el punto de vista de cada individuo, la función a optimizar es dinámica (cambia a medida que cambia el resto de la población).
- El concepto de convergencia del algoritmo cambia; no consiste en que todos los individuos se aproximen entre sí, sino que la población del algoritmo debe permanecer diversa. Requiere por lo tanto mecanismos de diversificación.
- La solución del problema es una parte del conjunto de individuos. Esto quiere decir que, para evaluar el éxito del algoritmo, es preciso realizar un proceso de selección entre los individuos para extraer aquellos que deben formar parte de la solución.

En la definición del algoritmo PSC se cuidará que el posicionamiento de los prototipos atienda al criterio de preservación de la frontera, a diferencia de la mayor parte de los algoritmos de clasificación por prototipo más próximo, que intentan localizar puntos próximos a los centros de los clusters.

Las modificaciones fundamentales que se introducen en el nuevo algoritmo, respecto al algoritmo PSO de partida, son los siguientes:

1. Las partículas utilizan una función de evaluación local para medir su éxito . Esta *función de fitness local* depende en cada momento de la posición del resto de las partículas, dado que el movimiento de una partícula vecino puede variar de forma importante los patrones que se incluyen en la región de Voronoi de la partícula considerada.

2. Las partículas influyen en su *vecindario local*, entendido éste como las partículas más próximas dentro del espacio de búsqueda. Junto con la función de fitness usada, el vecindario local genera los nichos necesarios para que la búsqueda se realice en torno a varios óptimos simultáneamente.

3. Las partículas interaccionan tanto mediante cooperación como mediante competición; la cooperación se modela como fuerza atractiva y la competición como fuerza repulsiva. La fuerza repulsiva actúa como mecanismo de diversificación.

A diferencia del algoritmo PSO, el enjambre de prototipos no tiene por qué ser homogéneo: las partículas tienen atributos, entre ellos la clase que asignan. Incluso esta individualización de cada uno de los prototipos nos ha permitido estudiar ciertas variantes de interés: la posibilidad de aplicar distintas medidas de proximidad local o la de adaptar el movimiento de cada prototipo en función de su importancia para la solución.

Con el fin de comparar ambas aproximaciones, hemos realizado experimentos con dos implementaciones particulares de las mismas, que describimos en las secciones siguientes.

3.2 Resolución mediante un Enjambre de Pittsburgh

Esta aproximación consiste en la resolución del problema de obtención de prototipos mediante el algoritmo PSO estándar. Para ello, se codifica un conjunto de prototipos en cada partícula. El problema de clasificación se transforma en un problema de optimización al utilizar como medida de fitness el error de clasificación del conjunto de prototipos sobre el conjunto de entrenamiento.

3.2.1. Codificación de la solución

Como se ha indicado, el algoritmo PSO utiliza una población de partículas, cada una de las cuales codifica una solución completa de un problema de optimización. En nuestro caso una partícula codifica un conjunto de prototipos. Cada prototipo es un punto del espacio definido por los atributos del problema de clasificación.

Cada prototipo debe tener asociada, además, la clase predicha por el mismo. Existen varias formas de atribuir un valor de clase a un prototipo:

- Asociar una clase a cada prototipo, que varía con el tiempo de forma evolutiva, como si fuera un atributo más de cada uno de los prototipos codificados en la partícula (clase dinámica).
- Asociar a cada prototipo la clase correspondiente a la mayoría de los patrones que existan en su región de Voronoi asociada (clase por votación).
- Asociar de manera estática una clase a cada prototipo, de forma que no varía durante la ejecución del algoritmo (clase estática). En este caso no es preciso utilizar ninguna codificación para las clases, puesto que la posición dentro de la partícula puede usarse para determinar la clase del prototipo.

En nuestra implementación descartamos la posibilidad de usar clases dinámicas ya que se utiliza la versión continua de PSO, que no admite atributos discretos. Por otro lado, se experimentó con un sistema de clase por votación, pero se observó que tenía dificultades para generar prototipos para clases con pocos patrones: al realizar la votación, dichas clases se descartan en beneficio de las clases más numerosas. Por dicho motivo utilizamos finalmente clases estáticas.

En una codificación de tipo Pittsburgh, es preciso definir el número de prototipos que se codifica en cada partícula. Si las clases son estáticas, en lugar del número total de prototipos podemos hablar del número de prototipos por clase. El caso más sencillo sería el utilizado en [Falco et al., 2006], ya comentado.

A diferencia del caso anterior, en nuestra implementación permitimos un número n de prototipos por cada clase. Si el número de clases es k, cada partícula codifica $n \cdot k$ prototipos. La estructura de la partícula resultante

es la de la Tabla 3.1. En dicha representación, asumimos que cada prototipo está definido por d atributos numéricos (del atributo 1 al atributo d). Numeramos las clases de la partícula con los índices 1 a k. El primer prototipo corresponde a la clase 1, el segundo a la clase 2, etc. hasta k, y la secuencia se repite en total n veces.

Es destacable que el número máximo de prototipos en la solución ($n \cdot k$) determina la dimensión de la partícula, y por lo tanto la dimensión del espacio de búsqueda del algoritmo.

La dimensión total de la posición de la partícula es por lo tanto $n \cdot d \cdot k$.

Tabla 3.1 Codificación de un conjunto de prototipos en una partícula, para un enjambre tipo Pittsburgh. d: Número de atributos ; k: Número de clases; n: Número de prototipos por clase.

	Prototipo 1	...	Prototipo k	...	Prototipo $n \cdot k$
Posición	$x_{1,1}\ x_{1,2}\ \ldots x_{1,d}$		$x_{k,1}\ x_{k,2}\ \ldots x_{k,d}$		$x_{n\cdot k,1}\ x_{n\cdot k,2}\ \ldots x_{n\cdot k,d}$
Clase	1	...	k	...	k

Planteado en estos términos, el algoritmo PSO estándar puede ejecutarse para encontrar la partícula que optimice la función de fitness que se defina. Al usar como función de fitness la tasa de éxito de clasificación, la solución del algoritmo aproxima una solución del problema de reemplazo de prototipos que definimos inicialmente.

3.2.2. Pseudocódigo de PSO con aproximación de Pittsburgh

Describiremos el algoritmo de clasificación con PSO de Pittsburgh usando los símbolos utilizados en la Tabla 3.1 y en las Ecuaciones 2.17 y 2.13. Es conveniente resaltar que, con la definición que hacemos del fitness de las partículas, nuestro algoritmo es un algoritmo de *maximización*. Por lo demás, se trata del algoritmo estándar PSO, particularizado con la notación que usamos en este documento.

Una partícula (Pob_i) se define como un conjunto de vectores y un valor de fitness, tal y como se muestra en la Ecuación 3.1. Los vectores tienen la misma dimensión, $n \cdot d \cdot k$ según la notación de la Tabla 3.1.

$$Pob_i = \{\mathbf{x}_i, \mathbf{v}_i, \mathbf{p}_i, MejorFitnessP_i\} \quad (3.1)$$

El algoritmo comienza inicializando una población de partículas (Pob). El tamaño de dicha población es un parámetro del algoritmo. La posición ($\mathbf{x}_i$) comienza como un vector aleatorio dentro de los límites del espacio de atributos; igualmente la velocidad ($\mathbf{v}_i$) se inicializa de forma aleatoria. Cada partícula incluye el valor de fitness de la mejor posición obtenida hasta el momento ($MejorFitnessP_i$), que empieza con valor 0.

Se inicializa una variable para todo el enjambre que guardará el mejor fitness ($MejorFitnessEnjambre$) y la solución provisional es un partícula que queda vacía ($MejorParticula$). Dicha solución provisional se actualizará a partir de la primera iteración del algoritmo.

Además, debe definirse el vecindario de cada partícula ($Vecindario_i$), para el que hemos escogido una topología en anillo: la partícula 2 es vecina de la 1 y la 3, la partícula 3, vecina de la 2 y la 4, y así sucesivamente.

A continuación, se itera el algoritmo hasta que se alcance la condición de parada (tasa de éxito del 100 %) o se alcance un número de iteraciones máximo ($MaxIter$). En cada iteración, se realiza la siguiente secuencia de operaciones para cada partícula:

1. Clasifica el conjunto de entrenamiento E con los prototipos de la partícula.
2. Calcula el Fitness de la partícula como el éxito de clasificación obtenido.
3. En caso necesario, actualiza las variables que guardan la mejor posición de la partícula hasta el momento ($\mathbf{p}_i$) y su fitness ($MejorFitnessP_i$).
4. En caso necesario, actualiza las variables que guardan la mejor partícula obtenida hasta el momento ($MejorParticula$) y su valor de fitness correspondiente ($MejorFitnessEnjambre$).
5. Calcula cuál es el "mejor vecino" ($\mathbf{p}_{\mathbf{g}i}$) de la partícula, cuya posición forma parte de las expresiones de movimiento del enjambre.
6. Actualiza la velocidad de la partícula ($\mathbf{v}_i$) según la fórmula de PSO.

Una vez calculada la velocidad para cada partícula, actualiza la posición de todas las partículas para la iteración siguiente y se repite el ciclo.

Al finalizar el algoritmo, se utiliza la mejor solución encontrada, almacenada como la posición de la $MejorParticula$, para clasificar el conjunto de validación V. La tasa de éxito en este conjunto mide el resultado del algoritmo.

La secuencia de este algoritmo se describe de forma esquemática en el Algoritmo 1.

3.2.3. Función de fitness en la aproximación de Pittsburgh

La función de fitness que se usa para evaluar una partícula es simplemente el porcentaje de éxito de clasificación, calculado según la Ecuación 3.2, donde $|G|$ es el número de patrones bien clasificados y $|E|$ el número total de patrones de entrenamiento.

$$\text{Fitness} = \frac{|G|}{|E|} \cdot 100 \tag{3.2}$$

Para calcular la tasa de éxito final del algoritmo, se utiliza la misma expresión pero sobre la clasificación que realizan los prototipos de la mejor partícula sobre un conjunto de validación V, como muestra la Ecuación 3.3.

$$\text{Resultado} = \frac{|G|}{|V|} \cdot 100 \tag{3.3}$$

3.2.4. Ecuaciones de movimiento

Utilizamos en este caso el algoritmo PSO estándar tal y como se define en [Bratton and Kennedy, 2007]. En dicho trabajo se resumen las que se consideran mejores prácticas en la optimización mediante PSO, en cuanto la elección de los valores de los parámetros y topología de vecindad.

El movimiento de la partícula viene determinado por las Ecuaciones 2.17 y 2.13. El vecindario que utilizamos es l_{best} estático con tres vecinos. Los parámetros restantes se detallan en el apartado de experimentación.

Algoritmo 1 Algoritmo de clasificación mediante PSO de Pittsburgh

```
1: E ← Patrones entrenamiento
2: V ← Patrones validación
3: Inicializa enjambre vacío (Pob = ∅)
4: MejorFitnessEnjambre ← 0
5: Iter ← 0
6: for i = 0 to |Pob| do
7:    Inicializa la posición de la partícula i (x_i ← Vector Aleatorio).
8:    Inicializa la velocidad de la partícula i (v_i ← Vector Aleatorio).
9:    Inicializa el mejor fitness de la partícula (MejorFitnessP_i ← 0).
10: end for
11: while MejorFitnessEnjambre < 100 % y Iter < MaxIter do
12:    for Pob_i ∈ Pob do
13:       Clasifica E con los prototipos codificados en Pob_i
14:       Fitness_i ← Tasa de éxito sobre E (ver 3.2.3)
15:       if Fitness_i > MejorFitnessP_i then
16:          MejorFitnessP_i ← Fitness_i
17:          p_i ← x_i
18:          if Fitness_i > MejorFitnessEnjambre then
19:             MejorFitnessEnjambre ← Fitness_i
20:             MejorParticula ← Pob_i
21:          end if
22:       end if
23:       p_gi ← x_j, siendo la partícula j ∈ Vecindario_i la de mejor
          MejorFitnessP_j.
24:       Actualiza la velocidad de la partícula v_i usando la Ecuación 2.17.
25:    end for
26:    for Pob_i ∈ Pob do
27:       Actualiza la posición de la partícula Pob_i (x_i) usando la Ecua-
          ción 2.13.
28:    end for
29:    Iter ← Iter + 1
30: end while
31: Solución ← MejorParticula
32: Clasifica V con los prototipos codificados en Solución.
33: Evalúa la tasa de éxito sobre V
```

3.3 Resolución mediante Enjambre de Prototipos

Describimos a continuación el algoritmo que denominamos Enjambre de Prototipos, resultado de aplicar un enfoque de Michigan en la codificación de las soluciones del problema ya descrito. La característica de un enfoque de Michigan es que cada individuo de la población representa sólo una parte de la solución del problema. Cuando hablamos de enjambres, esto quiere decir que cada partícula representa un solo prototipo.

Esta codificación implica que las reglas de movimiento del enjambre deben modificarse de modo que no tiendan a encontrar una única "mejor posición", que correspondería con la mejor posición posible para ubicar dicho prototipo. En este caso, las partículas deben moverse de manera que permanezcan dispersas por el espacio de búsqueda, localizando cada una un buen lugar para situar un prototipo. Como se ha indicado, ello supone que la función de fitness sea local, es decir, tenga en cuenta sólo la idoneidad de la partícula en su entorno próximo.

Con respecto a la clase, se plantean las mismas posibilidades que en el caso anterior: asignación dinámica, por votación, o estática. En nuestro caso, hemos seleccionado también la opción de "clase estática": es decir, cada prototipo tiene asignada una clase fija. La clase se implementa mediante un atributo adicional de la partícula.

La función de evaluación local tiene diversos óptimos; el algoritmo debe ser capaz de identificar todos ellos de forma simultánea, asociando un prototipo a cada uno. En este sentido, la versión de Michigan del algoritmo se comporta como un optimizador multimodal.

Sin embargo, no es posible utilizar un mecanismo multimodal clásico porque, al utilizar la regla del vecino más próximo, la función de fitness local se hace dinámica: en cada iteración, para una misma partícula, el valor de su función de fitness varía en función de las posiciones que hayan tomado las partículas restantes del enjambre. La intromisión de una partícula de una clase distinta puede variar drásticamente el valor de la función de fitness local en un punto.

Estas consideraciones llevan al diseño del algoritmo, que detallamos a continuación a partir del pseudocódigo, y luego precisando en apartados independientes los términos introducidos en éste.

Tabla 3.2 Codificación de un conjunto de prototipos en un Enjambre de Partículas de tipo Michigan (PSC)

	Posición	Clase
Partícula 1	$x_{1,1}\ x_{1,2}\ ...\ x_{1,d}$	1
Partícula 2	$x_{2,1}\ x_{2,2}\ ...\ x_{2,d}$	2
...	...	...
Partícula k	$x_{n,1}\ x_{n,2}\ ...\ x_{n,d}$	k
Partícula $k+1$	$x_{n+1,1}\ x_{n+1,2}\ ...\ x_{n+1,d}$	1
...	...	...
Partícula $2 \cdot k$	$x_{n+k,1}\ x_{n+k,2}\ ...\ x_{n+k,d}$	k
...	...	...
Partícula $n \cdot k$	$x_{n \cdot k,1}\ x_{n \cdot k,2}\ ...\ x_{n \cdot k,d}$	k

3.3.1. Codificación de la solución

Dado que la partícula contiene un solo prototipo, la codificación es directa: la partícula tiene d coordenadas, cada una correspondiente a un atributo del problema.

Si se desea representar una solución con n prototipos de cada una de las k clases del problema, el enjambre debe contener al menos $n \cdot k$ partículas. Es decir, la inclusión o eliminación de prototipos no varía la dimensión del espacio de búsqueda como ocurre en el enjambre con enfoque de Pittsburgh, sino el tamaño del enjambre.

En la Tabla 3.2 se representa dicho enjambre. Si se comparan con la codificación del enjambre de Pittsburgh (Tabla 3.1), se observa que la estructura de la posición de *una sola* partícula en el de Pittsburgh es análoga a la del enjambre completo en el Enjambre de Prototipos.

3.3.2. Pseudocódigo del Clasificador mediante Enjambre de Prototipos

Describiremos este algoritmo usando los símbolos utilizados en la Tabla 3.2 y en las Ecuaciones 3.11 y 2.13.

Las partículas siguen teniendo los mismos atributos relativos al fitness

($MejorFitnessLocal_i$) y se añade un atributo de clase ($Clase_i$), como se muestra en la Ecuación 3.4. La dimensión de todos los vectores es d.

$$Pob_i = \{\mathbf{x}_i, \mathbf{v}_i, \mathbf{p}_i, MejorFitnessLocal_i, Clase_i\} \tag{3.4}$$

La fase de inicialización es similar a la especificada en el Algoritmo 1. El tamaño de la población (Pob) sigue siendo un parámetro, que debe escogerse teniendo en cuenta el número máximo de prototipos que puede haber en la solución ($n \cdot k$). La clase de la partícula se asigna en esta fase; en los experimentos realizados se usa el mismo número de partículas para cada clase, si bien esta forma de inicialización puede modificarse con facilidad.

La mejor solución hasta el momento es un enjambre ($MejorEnjambre$) y su evaluación se guarda en $MejorEvalEnjambre$. Se inicializan estos valores con los que corresponden al enjambre inicial.

A continuación, se itera el algoritmo hasta que se alcance la condición de parada (tasa de éxito del 100 %) o se alcance un número de iteraciones máximo ($MaxIter$). En cada iteración, se realiza la siguiente secuencia de operaciones:

1. **En la versión con población adaptativa**, se generan partículas nuevas.
2. Para cada partícula,
   a) Calcula los vecindarios cooperativo (A_i) y competitivo (R_i) de la partícula
   b) Calcula el Fitness de la partícula ($FitnessLocal_i$).
   c) En caso necesario, actualiza las variables que guardan la mejor posición de la partícula hasta el momento ($\mathbf{p}_i$) y su fitness ($MejorFitnessLocal_i$).
   d) Calcula el factor Sf_i, que depende del $FitnessLocal_i$ de la partícula.
   e) Calcula cuáles son las posiciones de los vecinos cooperativo ($\mathbf{a}_i$) y competitivo ($\mathbf{r}_i$) de la partícula; estas posiciones forman parte de las expresiones de movimiento de PSC.
   f) Actualiza la velocidad de la partícula ($\mathbf{v}_i$) según la fórmula de PSC descrita más adelante.

3. Una vez calculada la velocidad para cada partícula, actualiza la posición de todas las partículas ($\mathbf{x}_i$).

4. A continuación, evalúa la tasa de éxito del nuevo enjambre sobre el conjunto E de entrenamiento, y, en caso necesario, actualiza las variables que guardan el mejor enjambre hasta el momento ($MejorEnjambre$) y su evaluación ($MejorEvalEnjambre$). A partir de aquí, se repite el ciclo para la iteración siguiente.

Al finalizar el algoritmo, se poda la mejor solución encontrada, almacenada como $MejorEnjambre$, mediante un algoritmo específico que retira las partículas del enjambre que perjudican la tasa de éxito. A continuación, la solución podada resultante se usa para clasificar el conjunto de validación V. La tasa de éxito en este conjunto mide el resultado del algoritmo.

El Algoritmo 2 muestra de forma esquemática esta secuencia. Los apartados referenciados describen cómo se calcula cada uno de los términos especificados en el mismo.

3.3.3. Fitness Local

En el Enjambre de Prototipos, cada partícula es un clasificador local, que debe evaluarse en función de su eficacia en su entorno próximo.

Como se ha visto al tratar el estado del arte (Apartado 2.1.1), existen diversas opciones para medir la calidad de un clasificador. En líneas generales, la mejor opción de calidad conjuga dos criterios: máxima precisión y máxima recuperación. Un clasificador (en nuestro caso una partícula) puede ser de buena calidad bien por no cometer errores, bien por realizar una clasificación correcta de un gran número de patrones.

En PSC agregamos ambos criterios en una función de fitness local de tipo jerárquico, que definimos a continuación.

Implementación en PSC

En el caso de PSC, tiene sentido definir como entorno local de una partícula el formado por su región de Voronoi del prototipo al que corresponde; es decir, la región del espacio donde están contenidos los patrones a los que dicho prototipo asigna su clase.

Algoritmo 2 Algoritmo de clasificación PSC

1: $E \leftarrow$ Patrones entrenamiento , $V \leftarrow$ Patrones validación
2: Inicializa enjambre vacío ($Pob = \emptyset$)
3: $Iter \leftarrow 0$
4: **for** $i = 0$ to $|Pob|$ **do**
5: Inicializa $\mathbf{x}_i$ y $\mathbf{v}_i$ de forma aleatoria.
6: Inicializa el mejor fitness de la partícula ($MejorFitnessLocal_i \leftarrow 0$).
7: Inicializa la clase de la partícula ($Clase_i \leftarrow i \mod k$).
8: **end for**
9: $MejorEnjambre \leftarrow Pob$
10: Clasifica E con las posiciones de las partículas como prototipos.
11: $MejorEvalEnjambre \leftarrow$ Tasa de éxito sobre E
12: **while** $MejorEvalEnjambre < 100\,\%$ y $Iter < MaxIter$ **do**
13: **(APSC sólo)** Reproducción de partículas (ver 3.4.1).
14: **for** $Pob_i \in Pob$ **do**
15: Calcula vecindarios cooperativo A_i y competitivo R_i (ver 3.3.5).
16: Calcula $FitnessLocal_i$ (ver 3.3.3).
17: **if** $FitnessLocal_i > MejorFitnessLocal_i$ **then**
18: $MejorFitnessLocal_i \leftarrow FitnessLocal_i$
19: $\mathbf{p}_i \leftarrow \mathbf{x}_i$
20: **end if**
21: Calcula Sf_i según se describe en el Apartado 3.3.6)
22: $\mathbf{a}_i \leftarrow \mathbf{x}_j$ $Pob_j \in A_i$ más próxima a Pob_i.
23: $\mathbf{r}_i \leftarrow \mathbf{x}_k$ $Pob_k \in R_i$ más próxima a Pob_i.
24: Actualiza velocidad $\mathbf{v}_i$ según la Ecuación 3.11.
25: **end for**
26: **for** $Pob_i \in Pob$ **do**
27: Actualiza la posición $\mathbf{x}_i$ según la Ecuación 2.13.
28: **end for**
29: Clasifica E con las nuevas posiciones de las partículas como prototipos.
30: $EvalEnjambre \leftarrow$ Tasa de éxito sobre E
31: **if** $EvalEnjambre > MejorEvalEnjambre$ **then**
32: $MejorEvalEnjambre \leftarrow EvalEnjambre$
33: $MejorEnjambre \leftarrow Pob$
34: **end if**
35: $Iter \leftarrow Iter + 1$
36: **end while**
37: *Solución* $\leftarrow Poda(MejorEnjambre)$ (ver 3.4.2)
38: Clasifica V con las posiciones de las partículas de la *Solución* como prototipos.
39: Evalúa la tasa de éxito sobre V.

Para las expresiones de la función de fitness, denominaremos G_i al conjunto de patrones correctamente clasificados por la partícula i (la clase de la partícula es igual que la clase deseada del patrón) y B_i al conjunto de patrones incorrectamente clasificados. Por último, P_i será el conjunto de patrones de entrenamiento que son de la misma clase que el prototipo i. Definimos asimismo $g_i = |G_i|$ y $b_i = |B_i|$

Los dos objetivos en la evaluación de una partícula pueden evaluarse con las siguientes expresiones:

- Una posibilidad es usar directamente la medida de precisión en la clasificación (Ecuación 3.5). Con ello se guía la búsqueda de manera que las regiones de Voronoi contengan sólo patrones de la misma clase que el prototipo.

$$\text{Precision}_i = \frac{|G_i|}{|G_i| + |B_i|} \tag{3.5}$$

  Si se desea una medida estrechamente relacionada con la precisión, pero que varíe entre -1.0 y 1.0, se podría usar la Ecuación 3.6.

$$\text{Precision}_i^{'} = \frac{|G_i| - |B_i|}{|G_i| + |B_i|} = 2 \cdot Pr - 1.0 \tag{3.6}$$

  Ambas funciones tienen una ventaja de cara a los principios que deseamos para el algoritmo, que es que cada prototipo puede evaluarse en función de información local.

- Un problema de usar simplemente la precisión como función de fitness es que todos los prototipos que sólo clasifican patrones de su clase tendrían igual valor de fitness (1.0). Parece preferible poder distinguir mediante su fitness los prototipos que clasifican correctamente muchos patrones de los que clasifican pocos. Para ello se puede usar una medida de recuperación (recall) (Ecuación 3.7).

$$\text{Recuperacion}_i = \frac{|G_i|}{|P_i|} \tag{3.7}$$

En PSC es preciso combinar estos dos objetivos en una función de Fitness Local que permita evaluar cada partícula. Esta combinación podría realizarse de diversas maneras, pero hemos optado por crear una función de

fitness jerarquizada, de modo que las partículas con precisión 1.0 se ordenan entre sí con un criterio de recuperación, pero manteniendo su valor de fitness siempre superior al de las partículas cuya precisión es inferior a 1.0. La función jerarquizada viene dada por la Ecuación 3.8.

$$\text{LocalFitness}_i = \begin{cases} \frac{g_i}{|P_i|} + 2.0 & \text{si } G_i \neq \emptyset \\ & \text{y } B_i = \emptyset \\ \frac{g_i - b_i}{g_i + b_i} + 1.0 & \text{si } B_i \neq \emptyset \\ 0 & \text{si } G_i = B_i = \emptyset \end{cases} \tag{3.8}$$

La función así definida es siempre positiva y tiene dos tramos distintos: los valores entre 0.0 y 2.0 identifican partículas que tienen algún patrón mal clasificado en su región de Voronoi y se ordenan por precisión; y los valores entre 2.0 y 3.0 son partículas que sólo tienen patrones bien clasificados y se ordenan por recuperación. La función de fitness local se representa de forma gráfica en la Figura 3.1.

Hemos de observar que esta función no se utiliza para evaluar el enjambre completo; para ello se usa simplemente la tasa de éxito global (Ecuación 3.2), una vez clasificados los patrones con la solución que devuelve el algoritmo.

Corrección mediante factores de distancia

Podemos observar que la Ecuación 3.8 no toma en consideración la distancia de la partícula a los patrones que clasifica. Al depender únicamente del número de patrones de cada clase que hay dentro de la región de Voronoi de la partícula, el paisaje de fitness presenta numerosas planicies. La evaluación de la partícula varía sólo cuando la partícula se mueve de forma que se aproxima o aleja de otra partícula vecina lo bastante como para que un patrón cambie de región.

Por ello, se ha corregido la función anterior introduciendo una medida de distancia entre la partícula y los patrones que clasifica. Esto se hace modificando la definición de los factores g_i y b_i, que pasa a ser la que se indica en las Ecuaciones 3.9 y 3.10. En éstas, el valor $d(\mathbf{x}_i, \mathbf{p})$ es la distancia Euclídea entre la posición de la partícula i y el patrón $\mathbf{p}$.

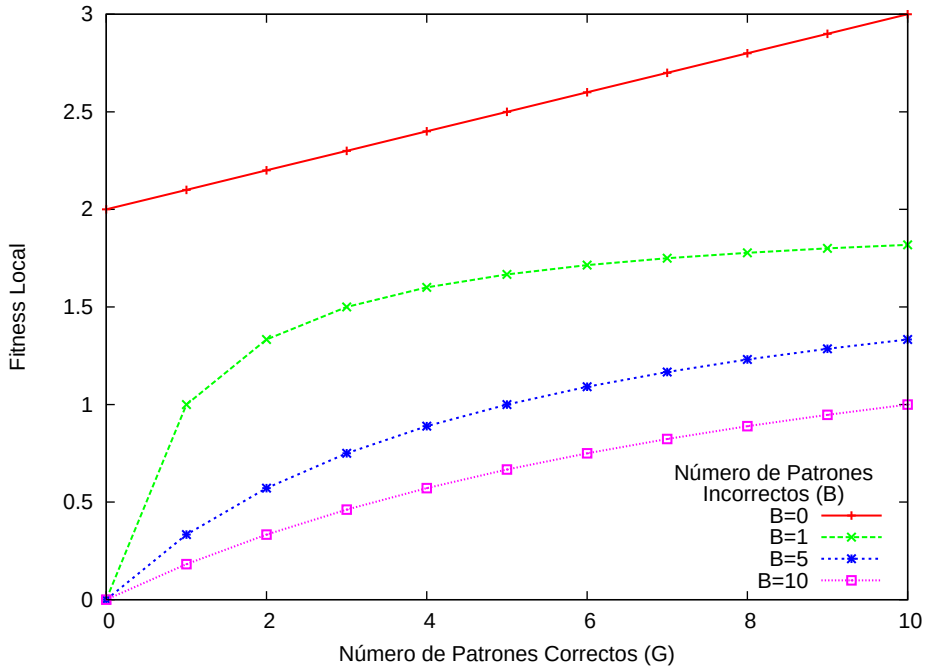

Figura 3.1 Función de Fitness Local. Gráficas variando el número de patrones correctamente clasificados, para distinto número de patrones incorrectamente clasificados ($B = \{0, 1, 5, 10\}$). Para el caso de $B = 0$ se toma como número total de patrones $|P_i| = 10$.

$$g_i = \sum_{\mathbf{p} \in G_i} \frac{1}{d(\mathbf{x}_i, \mathbf{p}) + 1.0} \tag{3.9}$$

$$b_i = \sum_{\mathbf{p} \in B_i} \frac{1}{d(\mathbf{x}_i, \mathbf{p}) + 1.0} \tag{3.10}$$

Al usar estos factores modificados, ambos se hacen de mayor valor en la medida en que los patrones a los que se refieren están más próximos a la partícula. Un patrón situado en la posición de la partícula contribuye con una unidad a la suma a la que corresponda (g_i o b_i); a medida que se aleja, la contribución a dicha suma disminuye.

En consecuencia, si comparamos con la versión anterior de la definición de los factores g_i y b_i, las partículas que están lejanas de sus patrones bien

clasificados y cercanas a sus patrones mal clasificados ven reducido su fitness. Ello permite generar un paisaje de fitness más rugoso que es más favorable para la búsqueda.

3.3.4. Ecuaciones de movimiento

La actualización de la velocidad de la partícula es el corazón de todo algoritmo de enjambre de partículas. En PSC, dicha función es sustancialmente diferente a la del algoritmo estándar, y viene dada por la Ecuación 3.11.

La ecuación expresa el valor para la iteración $t+1$ de la componente de velocidad de la partícula i para la dimensión d; se marca este hecho mediante los superíndices y subíndices correspondientes. Algunos de los términos de dicha ecuación se han descrito ya en el Apartado 3.3.2, pero resumimos aquí los factores que intervienen por conveniencia.

- Aparecen varios términos que forman parte de vectores asociados a la partícula: la velocidad anterior (v_{id}^t), así como su posición anterior (x_{id}^t), la mejor posición anterior de la partícula (p_{id}^t), y la del vecino cooperativo (a_{id}^t) y la del vecino competitivo (r_{id}^t).
- Los términos χ, w, y c_3 son parámetros constantes e iguales para todas las partículas y dimensiones.
- Los términos ψ_1, ψ_2, ψ_3, son factores aleatorios de una distribución uniforme entre 0 y 1, y proceden de los términos análogos de la ecuación del PSO estándar.
- El término Sf_i es el factor social adaptativo, que definiremos más adelante
- La función $sign()$ devuelve en valor $+1$, 0 o -1 según el argumento sea de signo positivo, cero, o de signo negativo, respectivamente.

$$\begin{aligned} v_{id}^{t+1} = \chi(w \cdot v_{id}^t + \psi_1 \cdot (p_{id}^t - x_{id}^t) \\ + Sf_i \cdot (\psi_2 \cdot sign(a_{id}^t - x_{id}^t) + c_3 \cdot \psi_3 \cdot sign(x_{id}^t - r_{id}^t))) \end{aligned} \tag{3.11}$$

Los términos llamados de "inercia" ($w \cdot v_{id}^t$) e "individual" ($\psi_1 \cdot (p_{id}^t - x_{id}^t)$) son exactamente iguales a los que se usan en el algoritmo PSO estándar.

Las diferencias con la ecuación estándar de PSO (Ecuación 2.17) están en los términos "sociales", es decir, los que expresan la interacción con las demás partículas: el término de atracción ($Sf_i \cdot \psi_2 \cdot sign(a_{id}^t - x_{id}^t)$) y el de repulsión ($Sf_i \cdot \psi_3 \cdot sign(x_{id}^t - r_{id}^t)$).

- Una de las diferencias básicas con PSO consiste en que se usa un vecindario dinámico y diferente según la clase de la partícula. La regla de selección de vecino que influye en el movimiento de la partícula es local, de manera que sólo influyan en la partícula partículas próximas en el espacio de búsqueda.
- El término de atracción supone que la partícula tienda a aproximarse a su vecino cooperativo (a_i^t). A diferencia de lo que ocurre en la ecuación del PSO estándar, esta influencia no depende de la distancia a la que ambas partículas se encuentren (compárese con la Ecuación 2.17). En pruebas preliminares se verificó que de esta forma la atracción es más gradual y el algoritmo obtiene mejor resultado.
- El término de repulsión no existe en la versión estándar de PSO. Es análogo al término de atracción, pero de signo distinto, de modo que la partícula tiende a alejarse de su vecino competitivo (r_i^t).
- Se introduce un coeficiente adaptativo (Sf_i), que afecta a los términos sociales y que varía con cada partícula y su historia de búsqueda; de este modo se puede regular la sensibilidad de cada partícula a la influencia de sus vecinos.

De igual manera que en PSO, la posición de la partícula en cada iteración se calcula mediante la suma vectorial de la velocidad a la posición anterior (Ecuación 3.12)

$$x_{id}^{t+1} = x_{id}^t + v_{id}^{t+1} \tag{3.12}$$

Los términos sociales (atractivo y repulsivo) se han diseñado para producir los siguientes comportamientos:

- Atracción: las partículas sólo se ven atraídas por otras que clasifican algunos patrones de forma incorrecta. Al atraer a su espacio partículas de las clases adecuadas, se espera que dichos patrones acaben siendo clasificados de forma correcta.

- Repulsión: se produce entre partículas de la misma clase, de forma que evita que varias partículas del mismo tipo se concentren en el centroide de los patrones de su clase más cercanos. En su lugar, algunas de las partículas tienden a moverse hacia los bordes de los agrupamientos de partículas de la misma clase. La posición del centro del cluster se ocupa si no hay partículas competitivas en el entorno.

3.3.5. Vecindario

La característica esencial del enjambre de Michigan es la localidad de la búsqueda. Las partículas no deben converger hacia una misma región del espacio. Ello se consigue definiendo la interacción entre partículas como una interacción local: es decir, sólo pueden interactuar las partículas que están próximas en el espacio de búsqueda. Esto requiere que el vecindario de la partícula se calcule de forma dinámica, comparando su posición con la de todas las partículas del enjambre en cada iteración.

Además, se ha definido un sesgo específico en la búsqueda de la posición de los prototipos: que éstos deben situarse preferiblemente próximos a la frontera de las regiones de Voronoi de clases distintas. Esto puede realizarse también mediante la definición del vecindario.

Para conjugar ambos criterios, el proceso de selección de los vecindarios y, con posterioridad, de los vecinos que intervienen en el movimiento de una partícula, se hace en dos fases:

Primero, se definen dos vecindarios diferentes para cada partícula, el *competitivo*, compuesto por las partículas que clasifican los mismos patrones y que deben tender a rechazarse; y el *cooperativo*, compuesto por las partículas que clasifican patrones de clase diferente, y que deben atraerse con el fin de tender hacia la frontera entre regiones de Voronoi.

- Para una partícula de clase C_i, su vecindario "cooperativo" (A_i) está compuesto por todas las partículas de clase distinta que contienen algún patrón de clase igual a C_i en su región de Voronoi (Ecuación 3.13). En esta expresión, B_j tiene el sentido definido ya en el Apartado 3.3.3; corresponde por lo tanto al conjunto de patrones mal clasificados por la partícula j.

$$A_i = \{j \in Pob \text{ , tal que } Clase_j \neq Clase_i \text{ y } B_j \neq \emptyset\} \qquad (3.13)$$

- Para una partícula de clase C_i, su vecindario "competitivo" (R_i) está compuesto por todas las partículas de igual clase que contienen algún patrón de dicha clase en su región de Voronoi (Ecuación 3.14). En esta expresión, G_j tiene el sentido definido ya en el Apartado 3.3.3; corresponde por lo tanto al conjunto de patrones bien clasificados por la partícula j.

$$R_i = \{j \in Pob \text{ , tal que } Clase_j = Clase_i \text{ y } G_j \neq \emptyset\} \qquad (3.14)$$

En la Figura 3.2 se muestra gráficamente un ejemplo en el que se indican los vecindarios cooperativo y competitivo, así como la fuerza de atracción y repulsión que aparece entre las partículas.

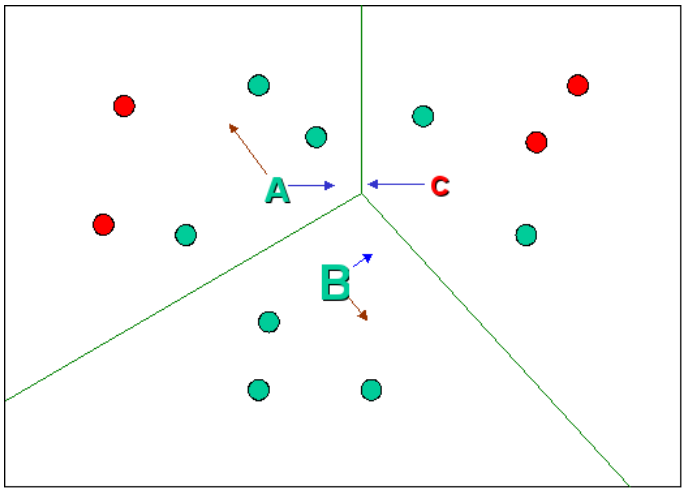

Figura 3.2 Vecindario e influencias sociales en PSC. Los patrones se representan mediante círculos y las clases mediante colores: "roja" o "verde". En esta situación: a) las partículas A y B se ven atraídas hacia C porque en su región de Voronoi hay patrones verdes; b) la partícula C se ve atraída hacia A porque en la región de A hay patrones rojos; y c) las partículas A y B se ven repelidas la una de la otra porque ambas clasifican patrones de clase "verde". Debe asumirse que A, B y C están alejadas del resto del enjambre y por lo tanto no tienen influencia de otras partículas.

Definidos los vecindarios, en cada instante t se seleccionan dentro de cada uno las partículas más cercanas (según la distancia Euclídea) a la partícula i. Así, la partícula escogida para atracción (a_i^t) es la partícula más próxima

a ella dentro de su vecindario cooperativo; y la partícula para repulsión (r_i^t) es la partícula más próxima dentro de su vecindario competitivo.

De esta forma, al utilizar las partículas más próximas se consigue que la influencia sea local, y que las partículas permanezcan distribuidas sobre el espacio de búsqueda.

Por otro lado, la fuerza de repulsión aleja las partículas de igual clase entre sí, y la de atracción reduce la distancia entre partículas de clase diferente. Mediante estos efectos se pretende que las partículas se sitúen próximas a los bordes de las regiones de clasificación.

3.3.6. Factor Social Adaptativo

Los términos sociales del algoritmo (atracción y repulsión) se ven afectados en las ecuaciones de movimiento (Ecuación 3.11) por un factor variable (Sf_i), asociado a la partícula de índice i.

La razón de su inclusión es intentar graduar la amplitud del movimiento de las partículas en función de su relevancia para la solución. Este factor se calcula en cada iteración del algoritmo.

Las partículas se mueven constantemente hacia su vecino "cooperativo" y se alejan de su vecino "competitivo". Si embargo, las partículas que ya están situadas de manera que representan una buena posición para un prototipo, deberían más bien moverse en el entorno de dicha posición para mejorarla. Es decir, para dichas partículas, el término de memoria debería ser más relevante que los términos sociales.

El factor Sf_i se puede visualizar fácilmente como un "peso" que depende inversamente del fitness de la partícula. El valor de fitness que se debe considerar es el mejor hasta el momento, ya que la posición de la partícula oscila en torno al mismo. Lo calculamos utilizado la (Ecuación 3.15). Se puede observar que las partículas que no clasifican ningún prototipo ignoran este término (su valor es 1), mientras que su valor se reduce hasta 1/3 en el caso de las mejores partículas; es decir, los factores sociales no ven reducida su influencia a cero en ningún caso.

$$Sf_i = \frac{1}{(MejorFitnessLocal_i + 1.0)} \tag{3.15}$$

Se realizó un conjunto de experimentos para ajustar este valor, cuyos

resultados se muestran en el Apartado 5.1.1.

3.4 Variantes del Algoritmo PSC

En esta sección detallaremos algunas variantes sobre el algoritmo PSC anteriormente descrito.

La primera, PSC con Población Adaptativa, pretende introducir un mecanismo por el que el número de prototipos, así como las clases de dichos prototipos, se ajusten de forma automática al problema de clasificación que el algoritmo aborda. Esto permite reducir la dependencia del algoritmo del tamaño de la población inicial, lo cual consideramos que es una característica deseable en algoritmos de reemplazo de prototipos. Este mecanismo se ejecuta "en línea" con la evolución del algoritmo PSC descrita, incluyendo una fase específica que se ejecuta en cada iteración del enjambre; en dicha fase, la población puede verse modificada mediante la inserción de nuevas partículas.

La segunda, PSC con Adaptación de Medida de Proximidad pretende mejorar el rendimiento del algoritmo, encontrando una medida de proximidad entre prototipos y patrones que no sea la distancia Euclídea y que no sea la misma medida para cada prototipo. Con ello el algoritmo debería ser más competitivo con algoritmos que utilizan selección de atributos, ponderación de atributos, o proyección del espacio de atributos. En esencia lo que se hace es variar la forma de los bordes de las regiones de Voronoi de cada prototipo, que dejan de ser lineales. Enfocamos esta variante como una optimización "a posteriori"; es decir, se ejecuta un algoritmo a continuación de la evolución de PSC, para mejorar sus resultados.

3.4.1. PSC con Población Adaptativa

Uno de los problemas que se presenta al intentar utilizar un algoritmo como el PSO estándar, es que no se puede cambiar la dimensión de la partícula, de modo que su capacidad de representación (el número de prototipos que puede codificarse en ella) está limitada. Dicho número, además, debe fijarse a priori, como parámetro del algoritmo.

En el caso de PSC, sin embargo, es sencillo adecuar el número de prototipos según las necesidades del problema, incluso de forma adaptativa. Basta con insertar más partículas en caso de que se considere que no son suficien-

tes, o eliminarlas si se considera que su presencia perjudica a la búsqueda. El criterio para realizar una u otra operación puede depender del fitness local.

Debe tenerse precaución para asegurar que la inclusión de un mecanismo como el considerado (inclusión o borrado de partículas) no perjudique en el caso en que la población inicial ya sea suficiente para representar la solución. De forma ideal, las partículas sólo deben introducirse cuando son necesarias, dado que un mayor número de partículas supone más complejidad, y por lo tanto degrada el rendimiento del algoritmo.

Las condiciones necesarias para el mecanismo de adaptación de la población son las siguientes:

- El número de partículas no debe crecer innecesariamente, perjudicando el coste computacional de ejecutar el algoritmo.
- El número de partículas no debe decrecer demasiado, reduciendo la diversidad de la población.
- La aparición de nuevos parámetros (tasa de reproducción o tasa de muerte) debe estudiarse y proponer valores base.

A continuación describimos la estrategia escogida para la reproducción de partículas. Cada partícula cuya región de Voronoi contiene patrones de varias clases tiene una probabilidad de generar partículas adicionales para todas las clases de dichos patrones. Estas partículas se colocarán en la posición de la partícula "madre" con velocidades aleatorias para que colaboren en la búsqueda a partir de la siguiente iteración.

- Sólo se consideran "elegibles" para reproducción las partículas que tienen algún patrón de clase diferente en su región de Voronoi ($B_i \neq \emptyset$).
- Para cada una de dichas partículas, se calcula la probabilidad de reproducción ($Prob_{rep}$) utilizando la Ecuación 3.17. Hemos decidido dar mayor probabilidad de reproducción a las partículas que tienen un valor de fitness local elevado, es decir, aquellas en cuyas regiones de Voronoi hay básicamente patrones de su propia clase ($G_i >> B_i$). Para ello se introduce el factor $MejorFitnessLocal_i$ en la fórmula, escalado al intervalo [0,1] mediante la Ecuación 3.16. La expresión de la probabilidad de reproducción viene dada por la Ecuación 3.17.

  En esta expresión hemos introducido un parámetro (λ_r) para ajustar el valor de esta probabilidad.

$$F_{norm} = \frac{MejorFitnessLocal_i - Min_{fit}}{Max_{fit} - Min_{fit}} \tag{3.16}$$

$$Prob_{rep} = \lambda_r \times F_{norm}^t \tag{3.17}$$

De forma complementaria, se realizó experimentación para incluir un mecanismo de borrado de partículas. Inicialmente se definió un criterio que pretendía descartar aquellas partículas cuyo fitness fuera especialmente bajo, de forma probabilística y análoga al mecanismo de reproducción. Sin embargo, la experimentación realizada llevó a determinar que este mecanismo limitaba la diversidad del enjambre, que en general obtenía soluciones con menos prototipos pero con peor resultado en clasificación. Por este motivo, dicho mecanismo no se ha incluido aún en esta variante del algoritmo. La definición de un proceso de eliminación de partículas que no presente este inconveniente ha quedado como desarrollo futuro para el algoritmo.

3.4.2. Algoritmo de Poda

Al final del proceso de evolución del enjambre, la solución obtenida ($MejorEnjambre$) puede mejorarse mediante un proceso de poda.

El algoritmo procede ordenando la población de la solución que se le proporciona por orden creciente de fitness. Se prueba a retirar una partícula, y se calcula de nuevo el porcentaje de éxito en clasificación sobre el conjunto de entrenamiento E. Si dicho porcentaje mejora, se continúa, para lo cual hay que volver a calcular el fitness local de las partículas, pues las regiones de Voronoi habrán cambiado; si no mejora, la partícula se reinserta y se continúa igualmente con el proceso en el punto en el que se encuentra.

El algoritmo de poda se describe a continuación de forma esquemática (Algoritmo 3).

3.4.3. Algoritmo de optimización local de medida de proximidad

La medida de proximidad que se utiliza en un algoritmo de clasificación por similitud puede determinar de manera importante el resultado del mismo. Sin embargo, no es posible determinar a priori, en un caso general, la

Algoritmo 3 Algoritmo de Poda de la solución

```
1: Elimina las partículas con MejorFitnessLocal_i = 0
2: Clasifica E con MejorEnjambre
3: Eval_b ← Tasa de éxito de clasificación sobre E
4: for i = 0 to |MejorEnjambre| do
5:    Retira la partícula de menor fitness local de MejorEnjambre
6:    Clasifica E con MejorEnjambre
7:    Eval ← Tasa de éxito de clasificación sobre E
8:    if Eval > Eval_b then
9:       Eval_b ← Eval
10:      Recalcula el Fitness local de las partículas restantes de
         MejorEnjambre
11:   else
12:      Inserta de nuevo la partícula i en MejorEnjambre
13:   end if
14: end for
15: Solución ← MejorEnjambre
```

medida que resultaría más adecuada. El caso habitual es utilizar la distancia Euclídea como se ha realizado hasta este momento.

Por otro lado, en líneas generales no tiene por qué existir una medida óptima para todo el espacio de atributos, sino que cabe la posibilidad de utilizar medidas diferentes en zonas distintas del mismo.

El propósito de esta sección es proponer una variante de nuestro algoritmo en la que cada prototipo utiliza una medida de proximidad diferente y posiblemente distinta de la Euclídea. La expresión para dicha medida se obtiene mediante el uso de un algoritmo evolutivo, de modo que se pueda adaptar a un conjunto amplio de problemas.

Si nos referimos a la notación que utilizamos en el Apartado 2.2.7, mediante esta variante pretendemos encontrar una función $w(f, \mathbf{p})$ que mejore el comportamiento del clasificador. La dependencia del prototipo considerado viene dada por la inclusión de la variable $\mathbf{p}$ en dicha expresión. En nuestro caso, usaremos una expresión dada por la Ecuación 3.18. Puesto que no se imponen restricciones sobre la matriz M, esta medida de proximidad no será en general una distancia matemática.

$$Prox_i(\mathbf{p}, \mathbf{q}) = \sqrt{(\mathbf{p} - \mathbf{q})^T \cdot M_{ip}^T \cdot M_{ip} \cdot (\mathbf{p} - \mathbf{q})} \qquad (3.18)$$

El problema viene definido por un conjunto de patrones de entrenamiento (E) y un conjunto (P) de prototipos a optimizar, que no varían durante la evolución de este algoritmo. El conjunto P es la solución que se obtiene mediante el algoritmo PSC o APSC ejecutado previamente sobre el mismo conjunto de patrones de entrenamiento E.

Cada individuo ($\mathbf{Pob}_i$) de la población (Pob) codifica una matriz de pesos completa para cada prototipo. Cada matriz está compuesta por un conjunto $d \times d$ de números reales, siendo d el número de atributos del problema. Por lo tanto, la dimensión de cada individuo es $d \times d \times |P|$ y el individuo i completo vendría dado por el vector de matrices de la Ecuación 3.19.

$$\mathbf{Pob}_i = (\mathbf{M}_{i1}, \mathbf{M}_{i2}, ..., \mathbf{M}_{i|P|}) \tag{3.19}$$

Puesto que los parámetros de la expresión anterior son matrices de números reales, hemos escogido un algoritmo de optimización continua. Se trata de las Estrategias Evolutivas [Bäck and Schwefel., 1992], [Bäck et al., 1991] con población y elitismo. En la versión que utilizamos de Estrategia Evolutiva, cada individuo incluye también un vector de varianzas, de la misma longitud que el vector que contiene los elementos de las matrices.

El algoritmo comienza inicializando cada individuo de la población con un vector de matrices unidad. La varianza inicial para todos los elementos del individuo es 1.0.

Para generar una nueva población (Pob^2), primero se escogen $|Pob|$ padres. Entre dichos padres, siempre se inserta el mejor individuo de la población anterior, para conservar la mejor solución. Esta técnica de elitismo garantiza que el algoritmo nunca empeora el resultado de partida. El resto se escoge mediante un mecanismo de torneo.

A continuación, cada padre genera un hijo mediante una función de mutación, y dicho hijo se agrega a la población. En Estrategias Evolutivas, se usa una función de mutación gaussiana, en la que las varianzas de las distribuciones correspondientes a cada posición del individuo también se mutan en cada iteración. Tras la mutación, la nueva población tiene el doble de individuos (los padres y los hijos). En la fase de selección se conserva sólo la mitad de estos.

Los nuevos individuos son evaluados y el mejor se retiene como solución temporal del algoritmo. Para la evaluación, se asigna a cada individuo de índice i su función de fitness de la siguiente forma:

1. Se clasifica el conjunto de entrenamiento E con el individuo (i). Para ello, se asigna a cada prototipo ($\mathbf{p}$) de P la matriz del vector $\mathbf{Pob}_i$ que le corresponde ($\mathbf{M}_{ip}$). Para calcular la proximidad entre dicho prototipo y un patrón ($\mathbf{q}$), se utiliza la Ecuación 3.18. La matriz unidad de la que se parte corresponde con la distancia Euclídea. Con esta medida de proximidad, se utiliza la regla del vecino más próximo para realizar la clasificación.

2. Se utiliza como fitness del individuo el porcentaje de éxito obtenido en clasificación, sobre el conjunto de entrenamiento.

La nueva población pasa a reemplazar a la antigua, y el proceso continúa por un número $Iter_{opt}$ de iteraciones, al final de las cuales el mejor individuo queda como solución.

El proceso se describe de forma detallada en el Algoritmo 4.

Algoritmo 4 Algoritmo de Optimización de Medida de Proximidad

$MejorEval \leftarrow$ Tasa de éxito de clasificación
for $i = 1$ to $|Pob|$ **do**
  $\mathbf{Pob}_i \leftarrow$ Matriz unidad para todos los prototipos
end for
for $i = 0$ to $Iter_{opt}$ **do**
  $\mathbf{Pob}_0^2 \leftarrow$ **MejorIndividuo** (Elitismo)
  for $i = 1$ to $|Pob|$ **do**
    Selecciona un elemento de Pob por torneo ($\mathbf{Pob}_j$)
    $\mathbf{Pob}_i^2 \leftarrow \mathbf{Pob}_j$
  end for
  for $i = 0$ to $|Pob^2|$ **do**
    Individuo $\leftarrow Muta(\mathbf{Pob}_i^2)$
    $Eval \leftarrow$ Evalúa **Individuo** (Tasa de éxito de clasificación)
    if $Eval > MejorEval$ **then**
      $MejorEval \leftarrow Eval$
      MejorIndividuo $\leftarrow$ **Individuo**
    end if
    Inserta **Individuo** en Pob^2
  end for
  $Pob \leftarrow Pob^2$ (Reemplaza la población)
end for
Solución $\leftarrow$ **MejorIndividuo**

4

Marco Experimental

En este capítulo detallamos el marco experimental en el cual realizamos la experimentación con los algoritmos que proponemos. Los apartados siguientes detallan:

- Los algoritmos utilizados como referencia para realizar comparaciones y los paquetes de software que implementan dichos algoritmos.
- Los conjuntos de datos utilizados en la experimentación.
- La metodología de experimentación y los parámetros utilizados.
- Los resultados numéricos de los experimentos realizados con los algoritmos de referencia.

4.1 Selección de Algoritmos de Referencia

Se ha pretendido utilizar un número elevado de algoritmos como base de comparación para los algoritmos que proponemos. De este modo, pretendemos tener una idea más clara de las características de nuestra propuesta.

Los algoritmos seleccionados para comparación incluyen los siguientes:

- Se utiliza como comparación la implementación del algoritmo de reemplazo de prototipos basado en PSO de estilo Pittsburgh que describimos en el Apartado 3.2.
- De la familia de clasificadores basados en vecino más próximo, se usan 1-NN, 3-NN y LVQ [Kohonen, 1995]. Se ha añadido el algoritmo ENPC

(Evolutionary Nearest Prototype Classifier) [Fernández and Isasi, 2004], que genera prototipos usando un algoritmo evolutivo con operadores específicos, y la versión con ponderación local de atributos que denotamos como ENPC$_d$ [Fernandez and Isasi, 2008].

- En cuanto a algoritmos de clasificación basados en otros paradigmas, se han seleccionado: C4.5 (árboles de clasificación), PART (reglas), Naive Bayes, SMO (una implementación de máquinas de soporte vectorial), Redes de Base Radial, GAssist-ADI [Bacardit and i Guiu, 2003] (evolución de reglas mediante algoritmos genéticos) y el algoritmo de clasificación mediante generación de reglas borrosas de Ishibuchi Learning (Fuzzy Rule Based Classifier System, FRBCS [Ishibuchi et al., 1999]).

Las implementaciones de los algoritmos anteriores se han obtenido en la herramienta WEKA (Waikato Environment for Knowledge Analysis - Entorno para Análisis del Conocimiento de la Universidad de Waikato) [Witten and Frank, 2005] salvo para GAssist y FRBCS, para los que se ha recurrido a la herramienta KEEL [Alcala-Fernández et al., 2008].

En los algoritmos ejecutados con el paquete Weka, se utilizan las implementaciones que forman parte de dicho paquete, puesto que en algún caso son una variante específica del algoritmo. En particular, se usa IBK-1 e IBK-3 como implementación del clasificador 1-NN y 3-NN respectivamente; J48 corresponde con la implementación de C4.5 en Weka; y SMO es la implementación correspondiente de un clasificador que utiliza una máquina de soporte vectorial (SVM).

4.2 Selección de Problemas Artificiales

Se han utilizado algunos conjuntos de datos generados artificialmente para comprobar el rendimiento del algoritmo y obtener una primera impresión del comportamiento del mismo al realizar análisis de sus diversos parámetros. Los dominios artificiales son bidimensionales, lo cual permite su representación gráfica. Los datos que caracterizan dichos problemas se muestran en la Tabla 4.1.

El problema Cluster consiste simplemente en un conjunto de cinco agrupamientos generados aleatoriamente y claramente separables. Este problema sólo se utiliza para obtener una representación gráfica del tipo de soluciones que encuentra el algoritmo. El conjunto de patrones se muestra en la

Tabla 4.1 Problemas artificiales utilizados en los experimentos

Nombre	Instancias	Atributos	Clases	Distribución de clases
Clusters	80	2	2	40 / 40
Diagonal	2000	2	2	1000 / 1000

Figura 4.1, en la que se aprecian con claridad los agrupamientos.

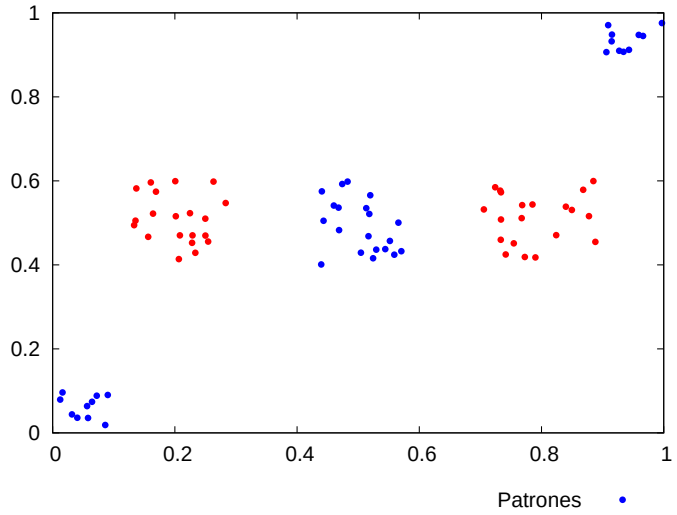

Figura 4.1 Patrones del problema "Cluster"; los patrones se agrupan en cinco clusters de dos clases, representadas con dos tonos distintos.

El problema Diagonal es otro problema bidimensional que es muy simple para un clasificador lineal. Se ha escogido porque la frontera de decisión entre las clases es evidente. El procedimiento para generar los patrones es el siguiente: se generan patrones aleatorios con coordenadas en [0,1], tales que si $x > y$, la clase es 1; y si $x \leq y$ la clase es 0.

En este caso, se utiliza parte de los patrones como conjunto de entrenamiento (un 25 %) y el resto se reserva para validación. El conjunto de patrones que queda para entrenamiento se representa en la Figura 4.2, en el que se puede observar que la diagonal del espacio de búsqueda (el rectángulo unidad) divide las regiones de cada clase.

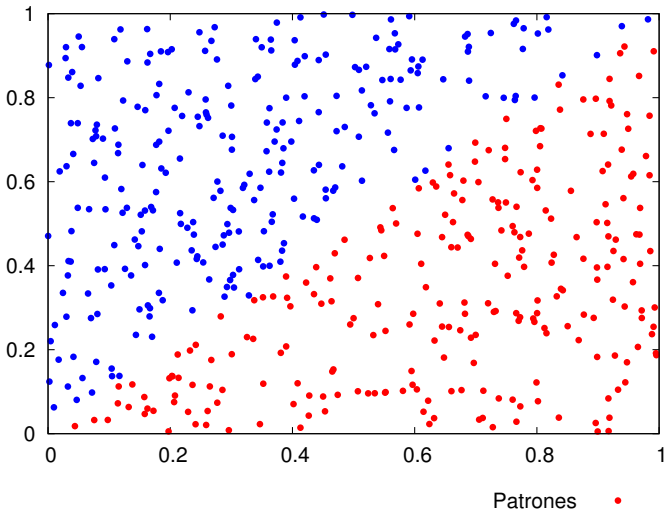

Figura 4.2 Patrones de entrenamiento del problema "Diagonal"; los patrones se distribuyen en dos regiones, una a cada lado de la diagonal. Las dos clases se representan con dos tonos distintos.

4.3 Selección de Problemas Reales

Se ha seleccionado un conjunto de dominios de clasificación a partir de la colección UCI [Asuncion and Newman, 2007] para evaluar los algoritmos sobre datos reales.

Se presenta un resumen de las características de estos dominios en la Tabla 4.2.

Al escoger estos dominios se ha procurado que fuera un conjunto diverso en cuanto al número de clases y atributos. Igualmente hay tanto dominios con clases balanceadas como no balanceadas.

4.3.1. Condiciones de la experimentación

Toda la experimentación se realiza a partir de los conjuntos de datos originales obtenidos en Internet, a los que se ha realizado un proceso de escalado lineal atributo a atributo. El resultado es que los valores de cada atributo se representan en el intervalo [0,1]. El escalado es diferente para cada atributo de entrada.

Tabla 4.2 Problemas reales utilizados en los experimentos

Problema	Patrones	Atributos	Clases	Distribución de clases
Balance Scale	625	4	3	288 / 49 / 288
Bupa	345	6	2	200 / 145
Diabetes	768	8	2	500 / 268
Glass	214	9	6	70 / 76 / 17 / 13 / 9 / 29
Iris	150	4	3	50 / 50 / 50
Thyroid (new)	215	5	3	150 / 35 / 30
Wisconsin	699	9	2	458 / 241

La forma de realizar la experimentación es idéntica en todos los casos. Para cada caso de experimentación, se ha probado el algoritmo 10 veces (10 experimentos); en cada uno de los experimentos, se ha dividido de forma diferente el conjunto de datos en segmentos de 10, y se ha realizado una ejecución del algoritmo. Es decir, el proceso ha sido, para cada **caso**:

1. Aleatorización del orden de los patrones
2. Diez *experimentos* con el algoritmo; en cada uno:
   a) División en diez **segmentos** para validación cruzada.
   b) Diez *ejecuciones* del algoritmo con cada segmento como conjunto de validación V, y el resto como conjunto de entrenamiento E.
   c) Generación de resultados medios del **experimento**, como media de las **ejecuciones**.
3. Obtención de estadísticos de mejor y peor **experimento**
4. Obtención de varianza entre **experimentos**
5. Obtención de estadísticos medios para todo el **caso**

Para cada conjunto de 10 experimentos que componen un caso, se han obtenido los estadísticos combinados (media, número de prototipos, número de evaluaciones, etc.). En total estas medias recogen 100 ejecuciones del algoritmo considerado (10 experimentos $\times$ 10 segmentos).

Tabla 4.3 Parámetros usados en los experimentos: P_c , número de partículas por clase; χ factor de constricción; w peso de inercia; c_1, peso del término cognitivo; c_2, peso del término atractivo; c_3, peso del término de repulsión; λ_r, tasa de reproducción

Algoritmo	Iteraciones	P_c	χ	w	c_1	c_2	c_3	λ_r
Pitt. PSO	300	-	1.0	0.72984	2.05	2.05	-	-
PSC (salvo Glass)	300	10	0.5	0.1	-	-	0.25	-
APSC (salvo Glass)	300	10	0.5	0.1	-	-	0.25	0.1
PSC (Glass)	500	20	0.15	0.1	-	-	0.25	-
APSC (Glass)	500	20	0.15	0.1	-	-	0.25	0.1

En algunas ocasiones se han proporcionado también los resultados del mejor y peor de los 10 experimentos. En estas ocasiones, primero es preciso obtener el resultado medio para cada experimento, calculando la media de los diez segmentos que lo componen; luego, se proporciona el mejor y peor resultado obtenido entre estas medias. Igualmente, cuando se proporciona la desviación típica del conjunto de experimentos, se trata de la desviación entre los valores obtenidos al agrupar todos los segmentos de cada experimento.

Se han realizado tests de significación estadística (prueba t de Student) entre los resultados en porcentaje de éxito sobre datos de validación, comparando los algoritmos que proponemos en este trabajo entre sí y con los algoritmos de referencia. Los resultados se muestran en las secciones correspondientes de los Capítulos 5, 6 y 7, indicando si la comparación es significativa, bien con un valor de certeza de un 95 % ($\alpha = 0.05$), o bien con un valor del 99 % ($\alpha = 0.01$).

También se proporcionan los resultados de dichos tests estadísticos en función de cada dominio, en los Apéndices (ver Apartado 9.1).

4.3.2. Resultados de referencia

En los capítulos relativos a experimentación (Capítulos 5, 6 y 7) se hace referencia a los resultados de la experimentación realizada con algoritmos de referencia. Con el fin de no reiterar dichos resultados, se proporcionan los resultados de dicha experimentación en las Tablas 4.4 y 4.5.

Los experimentos realizados con estos algoritmos utilizaron las herramientas que hemos detallado en el Apartado 4.1. En todos los casos se han realizado 10 experimentos de validación cruzada con 10 segmentos, de la forma indicada en el Apartado 4.3.1.

Tabla 4.4 Resultados de la experimentación con algoritmos de referencia basados en el vecino más próximo. En la parte superior, porcentaje de éxito de clasificación sobre datos de validación; en la parte inferior, desviación típica (en porcentaje)

Problema	Pitts. PSO	IBK (K=1)	IBK (K=3)	LVQ	ENPC	ENPC$_d$
Balance Scale	62.79	82.42	80.26	85.10	82.66	87.00
Bupa	65.13	62.22	62.48	62.18	59.41	64.15
Diabetes	74.54	70.40	73.89	74.26	68.42	73.54
Glass	52.52	69.07	69.05	62.56	68.48	64.24
Iris	90.89	95.40	95.20	95.60	95.33	95.07
Thyroid	88.93	96.93	94.31	91.03	94.81	94.81
Wisconsin	94.43	95.37	96.54	95.88	96.86	96.20

Problema	Pitts. PSO	IBK (K=1)	IBK (K=3)	LVQ	ENPC	ENPC$_d$
Balance Scale	1.46	1.15	1.48	2.13	1.14	0.47
Bupa	1.32	1.15	1.48	2.13	1.40	2.83
Diabetes	1.50	0.56	0.56	0.86	1.14	0.76
Glass	2.45	1.47	1.13	1.74	0.95	2.54
Iris	1.70	0.38	0.53	1.05	0.44	0.72
Thyroid	2.34	0.57	0.70	1.35	0.69	0.76
Wisconsin	0.56	0.17	0.30	0.28	0.28	0.23

Tabla 4.5 Resultados de la experimentación con algoritmos de referencia no basados en el vecino más próximo. En la parte superior, porcentaje de éxito de clasificación sobre datos de validación; en la parte inferior, desviación típica (en porcentaje)

Problema	J48	PART	Naive Bayes	SMO (SVM)	RBFNN	GAssist	FRBCS
Bal. Scale	77.82	83.17	90.53	87.62	86.25	81.81	78.22
Bupa	66.01	63.08	55.63	57.95	65.09	63.21	58.42
Diabetes	74.68	73.04	75.70	76.93	74.38	73.72	68.47
Glass	73.61	73.32	46.23	57.81	65.45	61.81	53.33
Iris	94.73	94.20	95.33	96.27	96.00	94.13	94.06
Thyroid	92.62	94.52	96.79	89.30	96.41	92.04	85.14
Wisconsin	94.59	94.61	96.07	96.74	96.08	96.03	95.40

Problema	J48	PART	Naive Bayes	SMO (SVM)	RBFNN	GAssist	FRBCS
Bal. Scale	0.68	0.76	0.30	0.55	0.81	2.34	2.09
Bupa	1.71	2.31	0.92	0.25	2.13	6.07	2.11
Diabetes	1.45	1.07	0.28	0.26	0.53	3.53	1.82
Glass	1.87	1.59	1.19	0.83	2.28	8.29	7.69
Iris	0.80	0.63	0.63	0.47	0.44	3.48	5.76
Thyroid	0.99	0.97	0.27	0.50	0.51	4.33	6.28
Wisconsin	0.41	0.42	0.10	0.18	0.24	2.15	2.04

5

Estudio Experimental de los Algoritmos PSC y APSC

En este capítulo se muestran los resultados de la experimentación realizada con el algoritmo PSC y su versión con población adaptativa, APSC.

En primer lugar, se utilizan algunos dominios cuyos datos se generan de forma artificial. La ejecución del algoritmo PSC sobre estos dominios se utiliza para confirmar la forma de funcionamiento del algoritmo y obtener una idea cualitativa de su evolución y forma de posicionamiento de los prototipos.

A continuación, se utilizan algunos de los dominios reales para estimar valores de los parámetros que puedan utilizarse de forma genérica en un conjunto amplio de dominios. Este estudio permite también comprender la influencia que dichos parámetros tienen sobre el rendimiento del algoritmo.

Se incluyen a continuación los resultados de PSC al realizar la experimentación diseñada sobre todos los dominios reales. Estos resultados se comparan con los algoritmos de referencia (ver Apartado 4.1) y se proporcionan los resultados de dicha comparación.

Por último, se realiza la experimentación correspondiente a la versión del algoritmo con población adaptativa (APSC), y se analizan dichos resultados de la misma forma que se hizo para PSC.

El capítulo concluye con una discusión sobre los resultados de ambos algoritmos en su comparación entre sí y con los métodos de clasificación de referencia.

5.1 Algoritmo PSC

En esta sección se detalla la experimentación realizada para evaluar el comportamiento del algoritmo PSC descrito en apartados anteriores. Se han realizado algunas pruebas preliminares destinadas a:

- Obtener valores adecuados para los parámetros del algoritmo.
- Describir el comportamiento del mismo, de forma cualitativa, utilizando para ello, sobre todo, los dominios artificiales, por ser bidimensionales.
- Evaluar cuantitativamente el resultado del algoritmo, realizando las comparaciones estadísticas con otros algoritmos de clasificación.

5.1.1. Estudio de parámetros

Al tratarse de un algoritmo nuevo, no se dispone a priori de información sobre el efecto de los parámetros en el resultado del algoritmo. Con el fin de tener alguna información sobre este particular, se han realizado una serie de pruebas preliminares, de modo que se pueda fijar un conjunto de parámetros que, en términos generales, proporcionen unos resultados satisfactorios en muchos dominios.

La experimentación preliminar se realizó con el dominio artificial Diagonal y con algunos de los dominios UCI: Bupa, Diabetes, Thyroid y Wisconsin.

De dicha experimentación se extrajeron las conclusiones que referimos a continuación. Los parámetros hacen referencia a la Ecuación 3.11:

- El parámetro w (factor de inercia), controla la influencia de la velocidad de la iteración anterior. Funciona como un término de "momento" que permite incrementar la exploración en algunos casos. Se realizaron experimentos con valores entre $w = 0$ y $w = 1.4$, y se llegó a la conclusión de que es preferible que se mantenga con un valor bajo. El motivo es el carácter dinámico de la función que se optimiza: en cada iteración, al cambiar las posiciones de las partículas, la función de fitness local varía. Por este motivo, las condiciones de la iteración pasada no deben influir en exceso en la iteración actual. Para la experimentación posterior, se fijó su valor en $w = 0.1$.

- El parámetro c_3 determina la importancia de la fuerza de repulsión con respecto al resto de las influencias que se ejercen sobre el movimiento de la partícula. En la Ecuación 3.11 los términos cognitivo y atractivo no tienen coeficiente. Con el fin de estimar el valor que debe utilizare para este parámetro, se experimentó con valores entre $c_3 = 0$ y $c_3 = 2.0$. Dicha experimentación determinó que la fuerza de repulsión debe ser de menor grado que la fuerza de atracción, ya que un exceso en este término produce la acumulación de las partículas cerca de los bordes del espacio de búsqueda. Se ha fijado un valor recomendado de $c_3 = 0.25$, que funciona correctamente para los dominios considerados.

- Se analizó también la variación del resultado en función del número de partículas en la población inicial. Para indicar este número usamos el número de partículas por clase (P_c). Se trata del mismo parámetro que en la descripción del algoritmo identificábamos como n (ver Apartado 3.3.1). El valor de este parámetro puede imponer una cota máxima al porcentaje de éxito, puesto que un número insuficiente de partículas puede no ser capaz de clasificar correctamente los patrones de un problema.

  Los resultados de la experimentación se muestran en la Tabla 5.1. Para los dominios considerados, se estimó que $P_c = 5$ resultaba insuficiente, mientras que al incrementar el número de partículas se observa una tendencia a mejorar el resultado. En la Figura 5.1 se muestra de forma gráfica el contenido de la tabla anterior, pero usando valores de tasa de éxito relativos al alcanzado para $P_c = 5$ (origen de coordenadas). Se observa que en los dominios Diagonal, Diabetes y Thyroid, el porcentaje de éxito parece estancarse a partir de $P_c = 20$; mientras que para Bupa y Thyroid no se observa dicho estancamiento.

Tabla 5.1 Experimentación preliminar: Éxito en clasificación (en %) al variar P_c

P_c	Diagonal	Diabetes	Bupa	Thyroid	Wisconsin
5	94.60	73.45	62.89	93.33	95.80
10	96.14	74.14	63.29	94.03	96.46
20	96.88	74.68	64.28	93.63	96.37
40	97.31	74.67	65.63	95.31	96.80

En la Tabla 5.2 se observa que el número de prototipos en la solución crece al aumentar P_c. También se observa que el número de prototi-

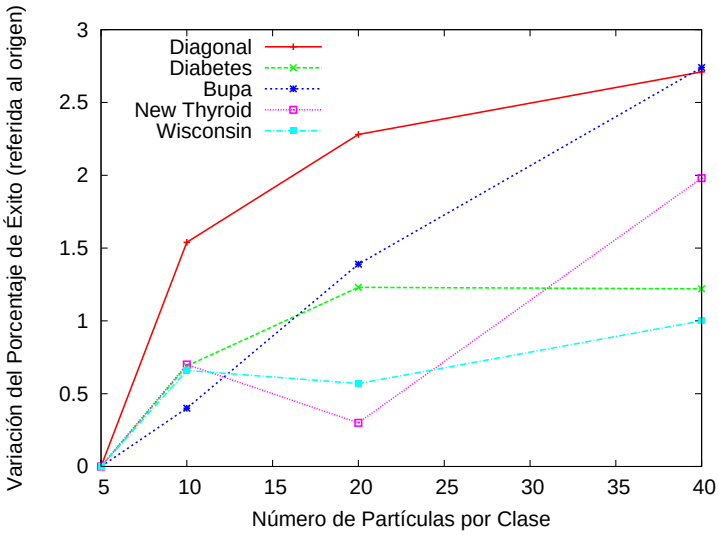

Figura 5.1 Variación del porcentaje de éxito al variar el número de partículas por clase (P_c). En horizontal, valores de P_c; en vertical, variación de la tasa de éxito respecto del caso inicial.

pos en la solución es inferior al producto $P_c \times K$ (siendo K el número de clases); ello se debe a que hay partículas que no se utilizan en la clasificación, al no ser las más próximas a ningún patrón. En la Figura 5.2 se observa que, en todos los dominios, la tendencia es creciente. Por otro lado, el crecimiento en P_c conlleva un crecimiento del coste computacional y tiempo de ejecución del algoritmo.

Al comparar con el estancamiento observado en tasa de éxito, concluimos que es conveniente mantener el valor de P_c reducido, pero que dicho valor puede influir en el resultado de forma importante; el valor de referencia que escogimos para todos los experimentos, salvo que se indique otro valor, fue $P_c = 10$

- Se estudió también el valor recomendable para el parámetro χ (*constriction factor*). Éste constituye un factor de "escala" que determina el máximo desplazamiento de una partícula en una iteración dada. Si el valor de χ crece, la posición de cada partícula puede variar con brusquedad entre dos iteraciones; es decir, el movimiento es más errático. Si se mantiene bajo, las partículas pueden optimizar mejor en el entorno local en el que se encuentran.

Tabla 5.2 Número medio total de prototipos en la solución (para todas las clases), al variar el número de prototipos por clase inicial, P_c.

P_c	Diagonal	Diabetes	Bupa	Thyroid	Wisconsin
5	8.55	6.39	6.48	7.74	6.03
10	17.39	10.47	10.89	11.37	11.12
20	33.45	17.10	17.44	16.73	21.54
40	57.67	29.50	29.08	22.08	40.63

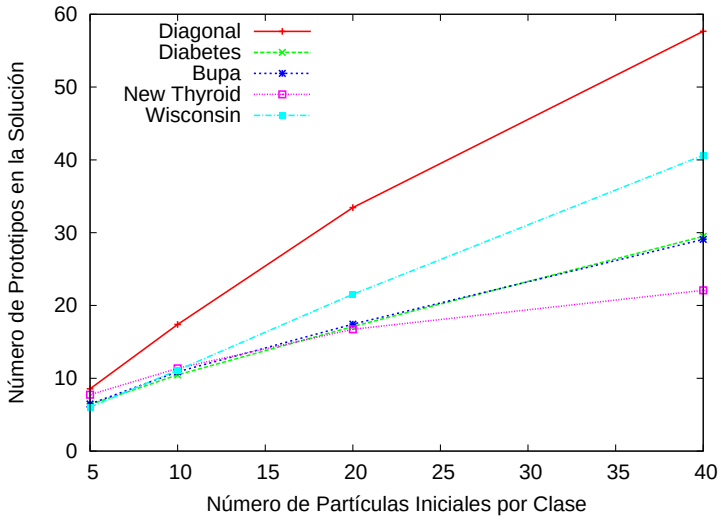

Figura 5.2 Variación del número de prototipos en la solución (P_c).

En la Tabla 5.3 se muestra la forma en que varía el porcentaje de éxito con este parámetro, cuando se fija $P_c = 10$. El porcentaje decrece al aumentar χ, siendo la variación es más importante en los dominios Bupa y Thyroid. Como se muestra en la Figura 5.3, el porcentaje decrece a partir del valor $\chi = 0.5$. Este valor se fijó como referencia en el resto de la experimentación. En la experimentación se detectó que, en algunos dominios, es preferible usar un valor de χ más reducido, que produce un movimiento de las partículas más lento; por ello, para el dominio Glass se utilizó un valor de $\chi = 0.15$.

Tabla 5.3 Éxito de clasificación (en %), al variar el coeficiente χ

χ	Diagonal	Diabetes	Bupa	Thyroid	Wisconsin
0.25	96.20	73.76	64.42	95.68	96.24
0.50	96.14	74.25	64.76	95.90	96.50
0.75	95.85	73.88	64.51	94.61	96.43
1.00	96.14	74.14	63.29	94.03	96.46

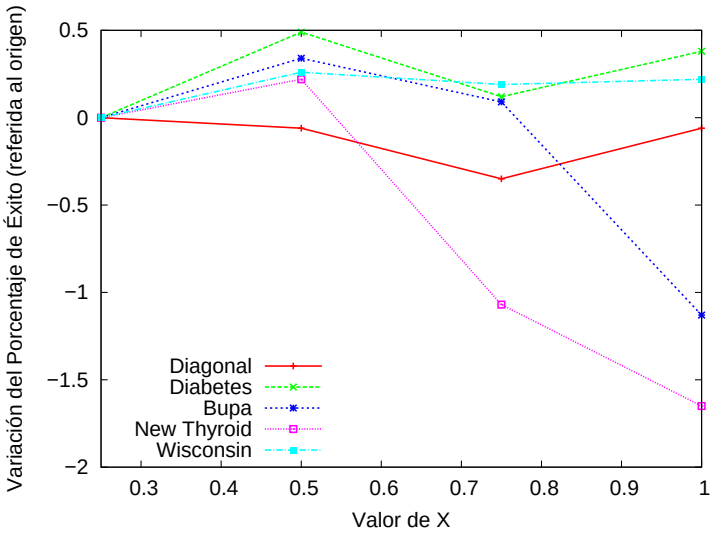

Figura 5.3 Variación del porcentaje de éxito al variar el parámetro χ, relativa al resultado obtenido con $\chi = 0.25$ partículas, en tanto por ciento.

5.1.2. Experimentación con problemas artificiales

Resultados

En la Figura 5.4 representamos ejemplos de las soluciones obtenidas para el problema "Cluster". Los patrones son de dos clases (identificados por el tono), clasificados por un enjambre con 16 partículas. Los patrones se muestran en trazo fino, y las partículas están coloreadas.

Se observa cómo las reglas de atracción y repulsión agrupan las partículas en torno a los diferentes clusters, pero también se muestra cómo las partículas de la misma clase se mantienen separadas por efecto de la fuer-

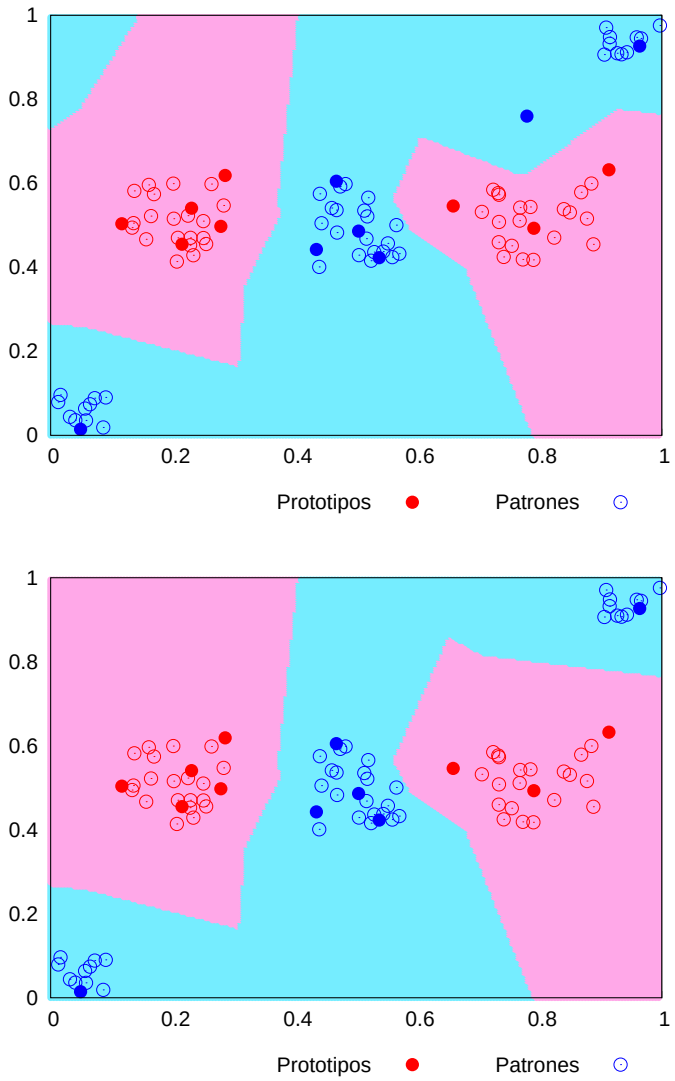

Figura 5.4 Ejemplo de solución para el problema "Cluster", antes y después del algoritmo de poda. Los patrones están en trazo fino, las partículas coloreadas. Se representan las regiones de Voronoi con tonos claros.

za de repulsión. Una vez el enjambre llega a una configuración de este tipo, donde todas las partículas clasifican patrones de la clase correcta, las fuerzas de atracción son nulas. Las fuerzas de repulsión no lo son, pero se compen-

san por efecto del término de memoria de la ecuación de velocidad. De ese modo las partículas pueden llegar a posiciones estables donde las fuerzas se encuentran equilibradas.

En la Figura 5.4 se observa también que, al detener la ejecución, puede haber partículas situadas en posiciones en las que no aportan nada a la clasificación, como la que se encuentra próxima a la posición $(0.8, 0.8)$ en la figura de la superior. De hecho, un prototipo situado en dicha posición introduce una distorsión al usar el clasificador, pues asume que en dicha zona debe asignarse su clase a patrones desconocidos. El algoritmo de poda (Apartado 3.4.2) se encarga de eliminar dichas partículas de la solución, como se muestra en la figura de la derecha, con las regiones trazadas una vez retiradas las partículas que no contribuyen a la clasificación.

En la Figura 5.5 se muestra un ejemplo de evolución del error de clasificación sobre el conjunto de entrenamiento a lo largo de las iteraciones del algoritmo, hasta la iteración 200. En este caso, el error de clasificación desciende hasta cero a partir de la iteración 150 y permanece en ese nivel. Con anterioridad a ese punto, el error de clasificación oscila. Ello es un reflejo de las oscilaciones de las partículas en torno a sus mejores posiciones, que pueden cambiar la evaluación del enjambre. La atracción hacia las mejores posiciones pasadas recupera la situación más favorable al cabo de algunas iteraciones.

También es posible observar que ya se encontró una solución con error nulo con anterioridad a la iteración 50, pero el enjambre no quedó detenido en este punto. Este hecho se debe a las componentes del movimiento que no se anulan por el hecho de que el error de clasificación global sea nulo. Este comportamiento es posible en el dominio Cluster porque las clases están claramente separadas.

En la Figura 5.6 se muestra una de las soluciones para el problema "Diagonal". Se representa una solución y los patrones de entrenamiento utilizados. Se puede observar cómo las partículas se desplazan hacia la diagonal que es la frontera entre las clases. Al ver el movimiento de las partículas en secuencia, se observaría cómo, si dos partículas de distinta clase quedan agrupadas de modo que son su centro de atracción recíproco, oscilan a ambos lados de la diagonal.

En la Tabla 5.4 se muestran los resultados obtenidos por PSC sobre los dominios artificiales. La tabla incluye los valores medios (para 100 experimentos) de: porcentaje de éxito en entrenamiento y validación (sólo pa-

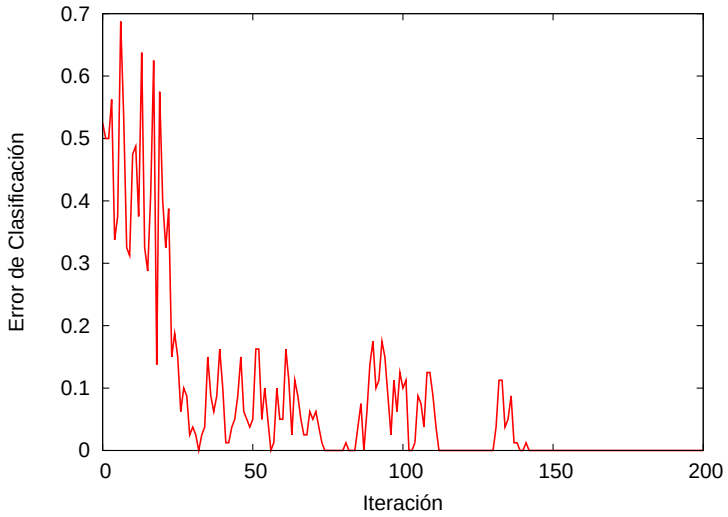

Figura 5.5 Ejemplo de evolución del error de entrenamiento para el problema "Cluster", en función de la iteración del algoritmo.

ra Diagonal), número de iteraciones realizadas hasta la mejor solución, y número de prototipos en la solución podada. Además, se incluye el porcentaje obtenido por el mejor experimento; este dato se refiere al conjunto de entrenamiento en el caso de Cluster, y al de validación en Diagonal.

En el dominio Cluster, se alcanza con facilidad el 100 % de acierto en la representación del conjunto de datos (sólo se usa entrenamiento). En Diagonal, el resultado es un poco peor, pero se debe a que la frontera teóricamente diagonal se convierte en aserrada debido a la utilización de sólo un pequeño número de patrones en torno a dicha diagonal, como se aprecia en la Figura 5.6.

Discusión de los resultados

Los resultados obtenidos sobre los dominios artificiales permiten comprobar algunas de las características del funcionamiento del algoritmo PSC, como son las siguientes:

- Las partículas localizan secciones diferentes del espacio de búsqueda y se agregan formando subenjambres según la región del espacio en la

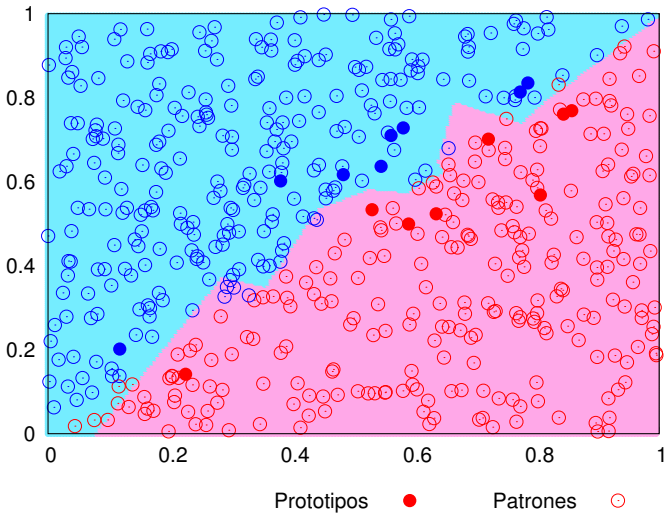

Figura 5.6 Ejemplo de solución para el problema "Diagonal", antes de la poda de la solución. Los patrones están en trazo fino, las partículas coloreadas. Se representan las regiones de Voronoi con tonos claros.

que se encuentren localizadas. Ello permite identificar clusters cuando estos se encuentran claramente separados.

- Como se comprueba en el dominio Diagonal, las partículas de clases distintas se atraen hacia las fronteras de decisión. Se observa que las partículas se alinean siguiendo los datos de entrenamiento de que disponen, y reproducen la frontera con mayor precisión en los lugares donde hay más partículas.

- El movimiento de las partículas puede hacerse más lento con el tiempo; sin embargo, si una partícula de una clase incorrecta "invade" el espacio de otras el porcentaje de éxito vuelve a oscilar, pero tiende a retornar a la posición anterior.

  Un ejemplo de esta situación lo podemos explicar sobre el problema Diagonal. Dos partículas de dos clases distintas, situadas a cada una en su lado de la frontera, pueden ser las que ejercen atracción entre sí, por estar alejadas de otras partículas. Si debido a dicha fuerza, sus posiciones se intercambian, el enjambre pasa a tener un error de clasificación elevado. En esas condiciones, tanto el factor de memoria (mejor posición anterior) como el factor de atracción se suman para

Tabla 5.4 Resultados de PSC sobre los dominios artificiales, para $\chi = 0.5$. Muestra porcentajes de éxito, número de evaluaciones hasta la mejor solución, número de prototipos y mejor experimento de los realizados.

Problema	Entrenamiento	Validación	Evaluaciones	Prototipos	Mejor Experimento
Cluster	100	N/A	378	12.21	100
Diagonal	97.24	96.14± 1.21	3453	15.04	96.73

atraer a las partículas hacia el lado correcto de la diagonal.

5.1.3. Experimentación con dominios de UCI

Resultados

En líneas generales el objetivo principal de la evaluación del algoritmo es obtener un buen resultado del clasificador en la clasificación del conjunto de validación. Sin embargo, al ser un algoritmo nuevo, resulta también de interés examinar los resultados del clasificador PSC sobre el conjunto de entrenamiento, para determinar en qué medida se ajusta al mismo. Realizamos esta comparación en la Tabla 5.5. Las columnas de dicha tabla contienen la información siguiente:

- Se muestra el porcentaje de éxito en entrenamiento y la desviación típica para los 10 experimentos completos (un total de 100 ejecuciones).
- El índice de representación mide el porcentaje que se pierde (por ello el signo negativo) al usar el conjunto de prototipos para clasificación, con respecto al 100 % que correspondería al clasificador 1-NN que se toma como referencia.
- Se muestra también el número de prototipos en la solución y el porcentaje de reducción respecto del número de patrones totales ($|E|$).
- Mostramos igualmente el porcentaje de éxito para el mejor y peor resultado entre los 10 experimentos de validación cruzada realizados.

Observamos que se han obtenido porcentajes de reducción en número de prototipos muy importantes. Estos porcentajes se deben al número reducido de partículas con las que se inicializa del enjambre. Ello justifica la pérdida de capacidad de representación, sobre el algoritmo 1-NN con el conjunto de entrenamiento completo.

Tabla 5.5 Resultados de PSC en entrenamiento. La desviación y mejor y peor resultado están calculados sobre experimentos completos (10 segmentos). Todos los datos en porcentaje, salvo el número de prototipos.

	Tasa de Éxito	Represen-tación	Proto-tipos	Reduc-ción	Peor Result.	Mejor Result.
Balance Scale	82.26 ±0.51	-17.74	12.90	97.71	81.60	83.01
Bupa	66.15 ±0.67	-33.85	11.75	96.22	64.83	66.92
Diabetes	74.44 ±0.27	-25.56	11.28	98.37	73.96	74.79
Glass	77.68 ±0.73	-22.32	47.52	75.38	76.45	78.92
Iris	97.30 ±0.56	-2.70	16.28	87.94	95.78	97.70
Thyroid	96.42 ±0.29	-3.58	16.33	91.58	95.89	96.85
Wisconsin	96.50 ±0.16	-3.50	11.99	98.09	96.27	96.80

En la Tabla 5.6, se muestra el porcentaje de éxito obtenido por PSC sobre datos de validación. Se muestran también los resultados de la versión de Pittsburgh de PSO y el resultado del test de significación estadística. El símbolo "+" significa que PSC es mejor que Pittsburgh PSO en el dominio indicado por la fila de la tabla, con un valor de certeza del 95 % ($\alpha = 0.05$). Si el símbolo aparece duplicado, la comparación es válida con un valor de certeza del 99 % ($\alpha = 0.01$). En negrita sobre fondo gris, los dominios en los que un algoritmo iguala el mejor resultado; la última fila suma el número de dominios en los que cada algoritmo obtuvo el mejor resultado.

Se observa que la versión de Pittsburgh sólo obtiene un resultado comparable a PSC para los dominios Bupa y Diabetes. En el resto de los dominios, PSC es significativamente mejor, con diferencias muy notables en muchos casos.

Otra característica que interesa comprobar es si el algoritmo es propenso al sobre ajuste. Dicha situación ocurriría en caso de que PSC obtuviera resultados más próximos al 100 % sobre el conjunto de entrenamiento, que probablemente redujeran su capacidad de generalización, es decir, su porcentaje de éxito sobre validación. Al comparar los porcentajes sobre validación

con los porcentajes de éxito sobre entrenamiento (Tablas 5.5 y 5.6), se observa que no se produce sobre ajuste, ya que los porcentajes en entrenamiento y validación son muy similares.

Tabla 5.6 Algoritmo PSC. Éxito en clasificación en validación y desviación típica (en %) y comparación con Pittsburgh PSO. El mejor resultado se marca en negrita sobre gris; los símbolos indican el resultado del test de significación (+ indica que PSC es significativamente mejor).

Problema	PSC	Pitts. PSO
Balance Scale	**82.59** ±0.96	62.79 (++)
Bupa	**64.76** ±2.31	**65.13** (=)
Diabetes	**74.25** ±1.18	**74.54** (=)
Glass	**74.23** ±1.76	52.52 (++)
Iris	**96.70** ±0.89	90.89 (++)
Thyroid	**95.90** ±0.87	88.93 (++)
Wisconsin	**96.50** ±0.26	94.43 (++)
Mejor	7	2

Comparación con otros algoritmos

En esta sección se muestran los resultados de los tests de significación estadística, realizados comparando el Algoritmo PSC con los algoritmos que usamos como referencia.

En la Tabla 5.7 se muestra el resultado de la comparación del porcentaje de éxito en validación de PSC con el resto de los algoritmos usados como referencia; en la parte superior, figuran los algoritmos que usan la regla del vecino más próximo; en la parte inferior, se muestra el resto. La notación es la indicada en leyenda. Los porcentajes de éxito de todos los algoritmos se pueden consultar en las Tablas 4.4 4.5.

Con respecto a los algoritmos basados en el vecino más próximo, podemos observar que los mejores resultados son de dos de los algoritmos de tipo evolutivo: PSC y ENPC$_d$. PSC está entre el grupo del algoritmo mejor en cinco del los dominios, mientras que ENPC lo está en 4. Algunos resultados de esta comparación son los que discutimos a continuación:

- Con respecto al algoritmo 1-NN básico, PSC obtiene resultados iguales o mejores en todos los dominios, excepto Thyroid, en el que 1-NN es el mejor de los algoritmos analizados. En términos generales, a pesar de que el algoritmo no representa con exactitud el conjunto de entrenamiento, sí tiene mejores características de generalización que 1-NN.

- PSC es capaz de mejorar el resultado de 3-NN sin necesidad de usar varios vecinos en su regla de decisión. La utilización del algoritmo 3-NN debe resultar ventajosa frente a 1-NN en casos en los que las fronteras óptimas de decisión no sean lineales, o en dominios en los que existan prototipos ruidosos, que generan clasificaciones incorrectas en el caso de 1-NN; estos patrones incorrectos afectan al resultado cuando se generaliza a prototipos desconocidos. Si embargo 3-NN no tiene ventaja sobre PSC en estos dominios.

- Observamos que LVQ no parece ser un buen algoritmo de selección de prototipos en estos dominios. Si se consulta la Tabla 4.4, puede verse que sólo está entre los mejores en el caso de Diabetes y Iris, y en ninguno de los dos supera a PSC. Ello podría apuntar a la hipótesis de que en estos casos, la mejor ubicación de prototipos no corresponde a la situación de los centroides sino que la estrategia de PSC es más adecuada.

- ENPC sólo iguala el resultado de PSC en Balance-Scale y Wisconsin. De los tests realizados entre algoritmos de referencia, se obtuvo que ENPC sólo obtiene un resultado igual al mejor algoritmo de los considerados en este último caso. No parece por lo tanto ser muy competitivo en estos dominios, si bien es posible que se deba a la existencia de problemas de sobreajuste. Se ha comprobado que el resultado en clasificación sobre el conjunto entrenamiento es muy elevado, y el número de prototipos generados se aproxima a los del conjunto original, en varios dominios. Por lo tanto, conviene tener en cuenta este hecho para utilizar una condición de parada del algoritmo basada en el porcentaje de éxito sobre conjunto de validación.

- ENPC_d resulta ser particularmente adecuado para el dominio Balance Scale, en los que es que obtiene el mejor resultado entre los algoritmos basado en el vecino más próximo. Obtiene resultados similares a los de PSC en todos los demás dominios excepto en Glass. Al comparar con la versión ENPC (sin ponderación de atributos), observamos que la

mejora considerablemente. Balance-Scale parece requerir un número elevado de prototipos para su representación. ENPC y ENPC$_d$ son capaces de ajustar el número de prototipos en la solución de forma adecuada a este dominio, mientras que PSC queda limitado por la inicialización.

Tabla 5.7 Comparación de PSC con algoritmos basados en el vecino más próximo (tabla superior) y resto de algoritmos (tabla inferior). El mejor resultado se marca en negrita sobre gris. Los símbolos indican el resultado del test de significación: "+" indica que PSC es mejor que el algoritmo indicado en la columna con $\alpha = 0.05$; "++" PSC es mejor con $\alpha = 0.01$; el signo negativo indica que el algoritmo de la columna es mejor que PSC.

	PSC	IBK-1	IBK-3	LVQ	ENPC	ENPC$_d$
Balance Scale	82.59	=	++	−−	=	−−
Bupa	**64.76**	++	+	++	++	=
Diabetes	**74.25**	++	=	=	++	=
Glass	**74.23**	++	++	++	++	++
Iris	**96.70**	+	+	=	+	+
Thyroid	95.90	−	++	++	+	=
Wisconsin	**96.50**	++	=	+	=	=
Mejor	5	1	2	2	1	4

	PSC	J48	PART	N.Bayes	SMO	RBFNN	GAssist	FRBCS
Bal. Scale	82.59	++	=	−−	−−	−−	=	++
Bupa	**64.76**	=	=	++	++	=	=	++
Diabetes	74.25	=	+	-	−−	=	=	
Glass	**74.23**	=	=	++	++	++	++	++
Iris	**96.70**	++	++	+	=	=	++	++
Thyroid	**95.90**	++	+	=	++	=	++	++
Wisconsin	**96.50**	++	++	=	=	=	=	++
Mejor	5	2	2	3	3	4	2	0

Con respecto al los algoritmos no basados en la regla del vecino más próximo, se observa que PSC y RBFNN son los algoritmos en más ocasiones igualan el resultado del mejor algoritmo. Por otro lado, en algunos dominios PSC obtiene resultados peores que Naive Bayes, SMO y RBFNN. Por último, PSC es el mejor algoritmo en solitario para el dominio Glass. En conjunto podemos afirmar que PSC es competitivo con el resto de los algoritmos en estos dominios.

Según los test realizados, Naive Bayes es el mejor algoritmo para Balance Scale, con diferencia sobre el resto, seguido de SMO. En Balance Scale es donde PSC obtiene resultados más pobres, siendo también superado por las Redes de Base Radial (RBFNN) y siendo igualado por PART. Como ya se observó con anterioridad, estimamos que el número de partículas con el que se ha inicializado PSC en este dominio resulta insuficiente.

5.2 Algoritmo PSC con Población Adaptativa (APSC)

En esta sección se muestran los resultados de la experimentación realizada con el algoritmo APSC, que incorpora a PSC un mecanismo de adaptación de la población.

Como se indica en el Capítulo 3, APSC incluye un proceso que detecta la conveniencia de introducir nuevas partículas en el enjambre para colaborar en la tarea de clasificación. El objetivo de APSC es reducir o eliminar la dependencia del parámetro de inicialización $|P_c|$, es decir, el número de partículas por cada clase con el que se inicializa el enjambre.

El mecanismo de adaptación se ha descrito en el Apartado 3.4.1, y requiere la introducción de un nuevo parámetro, la tasa de reproducción, λ_r.

5.2.1. Resultados

En la Tabla 5.8, se muestra el porcentaje de éxito obtenido por APSC sobre datos de validación. Se muestran también los resultados de PSC, así como el resultado de los tests de significación estadística en el que APSC se compara con PSC en cada uno de los dominios.

El formato de la tabla es el ya utilizado, donde el símbolo "+" significa que APSC es mejor que PSC en el dominio indicado por la fila de la tabla, con un valor de certeza del 95 % ($\alpha = 0.05$). Si el símbolo aparece dupli-

cado ("++"), la comparación es válida con un valor de certeza del 99 % ($\alpha = 0.01$). En negrita sobre fondo gris, se resaltan los dominios en los que un algoritmo iguala el mejor resultado; la última fila suma el número de dominios en los que cada algoritmo obtuvo el mejor resultado.

Se observa que, en cuatro de los dominios (Bupa, Iris, Thyroid y Wisconsin), la diferencia entre APSC y PSC no es significativa. Sin embargo, en tres de ellos (Diabetes, Balance Scale y Glass), la reproducción de partículas mejora significativamente el resultado. Es interesante resaltar que APSC no empeora nunca los resultados obtenidos por PSC en términos de porcentaje de éxito sobre el conjunto de entrenamiento. La mejora es más significativa en el dominio Balance Scale, donde el resultado de PSC era inferior en la comparación con otros algoritmos (ver Tabla 5.7).

Tabla 5.8 Algoritmo APSC. Éxito en clasificación en validación y desviación típica (en %) y comparación con PSC. El mejor resultado se marca en negrita sobre gris; los símbolos indican el resultado del test de significación (+ indica que APSC es significativamente mejor).

Problema	APSC	PSC
Balance Scale	**85.64** ±1.01	82.59 (++)
Bupa	**65.25** ±1.44	**64.76** (=)
Diabetes	**75.05** ±0.79	74.25 (+)
Glass	**76.70** ±2.54	74.23 (+)
Iris	**96.89** ±0.59	**96.70** (=)
Thyroid	**96.28** ±0.96	**95.90** (=)
Wisconsin	**96.51** ±0.59	**96.50** (=)
Mejor	7	4

En la Tabla 5.9 mostramos el número de prototipos que se encuentran en la solución de los distintos algoritmos. Observamos que es en Balance Scale y en Glass, seguido de Diabetes, donde se produce la mayor diferencia entre PSC y APSC. Se trata de los tres dominios en los que se produce una ganancia en porcentaje de éxito. Los resultados indican que APSC puede conseguir ganancia en resultado de la clasificación incrementando el número de prototipos, si bien esta ganancia no siempre se produce (Wisconsin).

El caso en el que el incremento es más importante es Balance-Scale, en que PSC no era un buen competidor con otros algoritmos, ni siquiera

con aquellos basados en el vecino más próximo. Ello confirma la hipótesis apuntada: la inicialización de PSC con 10 partículas por clase resultaba insuficiente para Balance Scale, y APSC es capaz de ajustar el número de prototipos de forma adecuada.

Con respecto al resto de los algoritmos que mencionamos en la Tabla 5.9, se puede observar que las soluciones obtenidas por la versión de Pittsburgh parecen estancarse en algunos casos, manteniendo un número de prototipos medio inferior a los de PSC. En Balance Scale, a pesar de no conseguir buenos resultados, la solución de Pittsburgh PSO tenía un número de prototipos elevado, lo cual apunta también a que la inicialización de PSC era insuficiente para este dominio.

Tabla 5.9 Número medio de prototipos en la solución , para Pittsburgh PSO, PSC y APSC

Problem	Pitts. PSO	PSC	APSC
Balance Scale	26.71	12.90	63.54
Bupa	8.48	11.75	15.04
Diabetes	13.00	11.28	18.49
Glass	11.40	47.92	55.01
Iris	9.82	16.28	17.65
Thyroid	7.19	16.33	19.66
Wisconsin	16.17	11.99	20.98

En la Tabla 5.10 se muestra el número medio de evaluaciones de la función de distancia necesario para obtener el resultado de cada uno de los algoritmos (Pittsburgh PSO, PSC y APSC). Son de destacar algunas observaciones:

- La versión de Pittsburgh siempre requiere un número de evaluaciones de la función de distancia mucho mayor; ello ocurre también en los casos en los que Pittsburgh PSO obtiene un resultado comparable al de PSC (ver Tabla 5.6). Esta diferencia es consecuencia de la codificación de la solución: en cada iteración de Pittsburgh PSO se realizan, para cada partícula, tantas evaluaciones de distancia como para todo el enjambre en PSC. Por lo tanto, en Pittsburgh PSO el número de evaluaciones de distancia queda aproximadamente multiplicado por el número de individuos de la población.

- En el dominio Iris, el algoritmo de Pittsburgh se estanca en valores bajos de porcentaje de éxito en momentos iniciales de su ejecución (ver Tabla 5.6). El hecho de que esto ocurra en el dominio más sencillo indica que el algoritmo se beneficiaría de algún mecanismo que permitiera salir de situaciones de estancamiento.

- De igual manera, en el dominio Glass, que es el más complejo en términos de número de atributos y clases, el algoritmo de Pittsburgh se estanca en una fase inicial de su evolución.

- Se observa también que la inclusión del mecanismo de reproducción de partículas en APSC lleva a un incremento en el coste computacional del algoritmo en todos los casos respecto a PSC. Este número mayor de evaluaciones se debe al incremento en el número de prototipos. Ello se ve compensado por el hecho de que, en algunos casos, APSC obtiene resulados significativamente mejores que PSC.

Tabla 5.10 Número de evaluaciones de distancia medios hasta la obtención del mejor resultado

Problema	Pitts. PSO	PSC	APSC
Balance Scale	99252	5783	24985.71
Bupa	64180	2827	3667.48
Diabetes	64636	3304.4	4968.05
Glass	2982	31490	44109
Iris	3026	4995	5640.25
Thyroid	103170	4955	5985.18
Wisconsin	65596	3575.6	5806.96

Comparación de APSC con otros algoritmos

En esta sección se muestran los resultados de los tests de significación estadística, realizados comparando el Algoritmo APSC con los algoritmos que usamos como referencia.

En la Tabla 5.11 se muestra el resultado de la comparación del porcentaje de éxito en validación de APSC con el resto de los algoritmos usados como referencia; en la parte superior, figuran los algoritmos que usan la regla del

vecino más próximo; en la parte inferior, se muestra el resto. La notación es la indicada en leyenda. Los porcentajes de éxito de todos los algoritmos de referencia se pueden consultar en las Tablas 4.4 4.5.

Con respecto a los algoritmos basados en el vecino más próximo, la inclusión de APSC mejora los resultados de PSC, pasando a ser el algoritmo que más veces obtiene el mejor resultado entre los algoritmos de vecino más próximo, con seis dominios sobre siete; en particular, al comparar con cada uno de estos algoritmos, podemos apuntar las siguientes observaciones:

- APSC pasa a igualar el resultado del mejor algoritmo (1-NN) en el dominio New Thyroid, donde el resultado de PSC era antes inferior al de 1-NN. En cuanto a la comparación con 3-NN, APSC mejora el resultado de éste en el dominio Diabetes (PSC sólo lo igualaba), y sigue sin presentar diferencia significativa en Wisconsin.
- Se observa también que APSC mejora los resultados frente a LVQ, que ya no lo supera en Balance Scale. En esta comparativa, LVQ pasa a ser el mejor algoritmo sólo en Diabetes.
- Por último, PSC obtenía igual resultado que ENPC en el dominio Balance Scale y Wisconsin. APSC mejora el resultado obtenido en Balance Scale frente a ENPC.

  Con respecto a la versión optimizada, ENPC_d, PSC era peor en Balance Scale y la diferencia no era significativa en Bupa, Diabetes, Thyroid y Wisconsin. APSC no consigue alcanzar el resultado de ENPC_d en Balance Scale, pero sí obtiene mejor resultado en Diabetes y Thyroid, y sigue siendo mejor en Glass e Iris. Por lo tanto, APSC pasa a ser un buen competidor de ENPC_d. ENPC_d sólo iguala en mejor resultado en tres de los dominios considerados.

Al comparar con ENPC, APSC mejora sus resultados en todos los dominios, excepto en Wisconsin, en el que los iguala. El efecto de la reproducción de partículas permite por lo tanto competir de forma ventajosa con este algoritmo evolutivo en estos dominios.

Con respecto a los algoritmos no basados en el vecino más próximo, observamos que el resultado es similar al obtenido en PSC (ver Tabla 5.7), si bien aparecen más situaciones en las que APSC mejora a otros algoritmos, o incrementa la significación del resultado; es decir, se incrementa el nivel de certeza del 95 % al 99 % ("+" pasa a "++").

5.3 Discusión de los Resultados

El algoritmo PSC permite obtener resultados comparables o mejores en porcentaje de éxito sobre el conjunto de validación que el resto de los algoritmos basados en la regla del vecino más próximo. Se puede consultar el Apartado 9.1 y la Tabla 9.8 para obtener una visión global de la comparación entre los resultados de los diversos algoritmos.

Esta buena capacidad de generalización se combina con un elevado índice de reducción: el resultado retiene del orden de un 90 % menos de información que el conjunto de entrenamiento en todos los dominios; en el caso de Glass la reducción está en el orden del 75 %. El algoritmo PSC no tiende al sobreentrenamiento, ya que los porcentajes de éxito sobre el conjunto de entrenamiento y el de validación son similares 5.5.

Sí se ha observado que el algoritmo puede no alcanzar un buen resultado si se subestima el número de partículas con el que se inicializa en enjambre, que influye en el número de prototipos en la solución.

Los resultados de la experimentación realizada con APSC confirman los resultados de PSC y añaden la posibilidad de que el algoritmo se adapte en cierta medida a un desajuste entre el número y distribución de prototipos por clase realizado en la inicialización y los requeridos por el problema considerado.

La introducción de APSC puede incrementar el número de prototipos en la solución sin mejorar la calidad del resultado, aumentando el coste computacional. Es necesario, por lo tanto, encontrar un compromiso entre el número de prototipos seleccionado en la inicialización y el uso de APSC.

APSC y PSC parecen adecuados para los dominios Bupa, Glass, Iris y Thyroid, en los que obtienen mejores resultados que los algoritmos de vecino más próximo. En estos casos podemos suponer que la forma de disponer los prototipos permite una mejor capacidad de generalización que la que ofrecen algoritmos de vecino más próximo de otras características (ver Tablas 5.7 y 5.11).

Por otro lado, APSC tiene más dificultad para superar el resultado que otros algoritmos en el dominio Wisconsin, que es el dominio con mayor número de atributos. Ello confirmaría la sensibilidad de APSC a la dimensión del espacio de atributos.

Por último, en Balance Scale el resultado no es totalmente satisfactorio,

puesto que no se iguala el resultado de ENPC_d y APSC queda bastante alejado del resultado de Naive Bayes. Este dominio parece cumplir la condición de distribución de probabilidad que permite que este clasificador destaque notablemente frente al resto.

Tabla 5.11 Comparación de APSC con algoritmos basados en el vecino más próximo (tabla superior) y resto de algoritmos (tabla inferior). El mejor resultado se marca en negrita sobre gris. Los símbolos indican el resultado del test de significación: "+" indica que APSC es mejor que el algoritmo indicado en la columna con $\alpha = 0.05$; "++" APSC es mejor con $\alpha = 0.01$; el signo negativo indica que el algoritmo de la columna es mejor que APSC.

	APSC	IBK-1	IBK-3	LVQ	ENPC	ENPC$_d$
Balance Scale	85.64	++	++	=	++	− −
Bupa	**65.25**	++	++	++	++	=
Diabetes	**75.05**	++	+	=	++	++
Glass	**76.70**	++	++	++	++	++
Iris	**96.89**	++	++	+	++	++
Thyroid	**96.28**	=	++	++	++	+
Wisconsin	**96.51**	++	=	+	=	=
Mejor	6	1	1	1	1	3

	APSC	J48	PART	N.Bayes	SMO	RBFNN	GAssist	FRBCS
Bal. Scale	85.64	++	++	− −	− −	=	++	++
Bupa	**65.25**	=	=	++	++	=	+	++
Diabetes	75.05	=	++	=	− −	=	++	++
Glass	**76.70**	++	++	++	++	++	++	++
Iris	**96.89**	++	++	++	=	=	++	++
Thyroid	**96.28**	++	++	=	++	=	++	++
Wisconsin	**96.51**	++	++	=	=	=	=	++
Mejor	5	1	1	3	3	4	1	0

6

Estudio de Sensibilidad al Ruido de APSC

Al tratarse APSC de un algoritmo que utiliza la regla del vecino más próximo, la presencia de ruido en los datos de entrenamiento puede afectar negativamente al comportamiento de este algoritmo. Por este motivo, se ha estimado conveniente realizar un breve estudio del comportamiento del algoritmo APSC en presencia de ruido de clase.

Con este objetivo, se ha partido de los mismos dominios procedentes de la colección UCI (ver Apartado 4.3), pero se han modificado variando el valor de la clase esperada para el 10 % de los patrones. Se ha repetido la experimentación anterior usando únicamente APSC, ya que su resultado siempre mejora o iguala a PSC. Los resultados se han comparado con los algoritmos utilizados en el Capítulo 5; se exceptúa GAssist, que no ofrecía resultados competitivos con APSC.

La hipótesis que deseamos comprobar es si la forma de posicionar los prototipos de APSC permite mejorar el comportamiento del algoritmo con respecto al de otros algoritmos de la misma familia, en presencia de datos de entrenamiento ruidosos. La utilización de prototipos puede mejorar el comportamiento respecto al ruido cuando se compara con el algoritmo de vecino más próximo (1-NN). El motivo es que la elección de un prototipo no suele realizarse utilizando la información de un solo prototipo sino de varios, de manera que un error en un patrón de entrenamiento no supone un error en el conjunto de prototipos generado.

6.1 Resultados

En la Tabla 6.1 se muestran los valores de éxito en clasificación obtenidos por los algoritmos de referencia basados en vecino más próximo, para los

conjuntos de datos con ruido; igualmente se muestran en la Tabla 6.2 los resultados para el resto de los algoritmos. En ambas tablas se incluye la desviación típica del resultado.

Tabla 6.1 Resultados de la experimentación con conjuntos de datos ruidosos, para los algoritmos de referencia basados en el vecino más próximo. En la parte superior, porcentaje de éxito de clasificación sobre datos de validación; en la parte inferior, desviación típica (en porcentaje)

	IBK.1	IBK-1	LVQ	ENPC	ENPC$_d$
Balance Scale	69.22	71.88	76.69	57.93	79.56
Bupa	59.01	60.27	58.44	57.06	57.21
Diabetes	60.52	65.36	68.16	58.25	68.75
Glass	56.25	59.98	57.32	58.10	57.14
Iris	75.00	81.60	82.07	71.73	83.13
Thyroid	79.21	83.42	79.51	74.86	84.14
Wisconsin	79.67	85.19	85.26	85.54	85.01

	IBK-1	IBK-3	LVQ	ENPC	ENPC$_d$
Balance Scale	0.50	0.61	1.14	1.02	0.63
Bupa	0.87	1.80	2.89	1.59	3.82
Diabetes	0.55	0.76	0.88	0.86	0.44
Glass	1.35	0.65	3.75	1.66	1.37
Iris	1.45	0.78	1.46	1.67	1.26
Thyroid	0.87	0.68	1.28	1.65	0.71
Wisconsin	0.59	0.19	0.47	0.67	0.35

En la Tabla 6.3 se muestra el resultado de la comparación del porcentaje de éxito en validación de APSC con el resto de los algoritmos usados como referencia; en la parte superior, figuran los algoritmos que usan la regla del vecino más próximo; en la parte inferior, se muestra el resto. La notación es la indicada en leyenda. Los porcentajes de éxito de todos los algoritmos de referencia se pueden consultar en las Tablas 6.1 6.2.

En las Tablas 6.1 y 6.3 se observa que APSC y ENPC$_d$ son los mejores algoritmos entre los basados en el vecino más próximo. Entre los restantes algoritmos, sólo LVQ obtiene un resultado igual para el dominio Diabetes.

Tabla 6.2 Resultados de la experimentación con conjuntos de datos ruidosos, para los algoritmos de referencia no basados en el vecino más próximo. En la parte superior, porcentaje de éxito de clasificación sobre datos de validación; en la parte inferior, desviación típica (en porcentaje)

Problema	J48	PART	Naive Bayes	SMO (SVM)	RBFNN
Bal. Scale	71.75	73.59	81.74	79.79	77.35
Bupa	61.83	61.46	55.09	57.30	60.31
Diabetes	67.00	67.72	69.94	72.15	68.57
Glass	63.60	61.51	44.56	48.90	56.21
Iris	80.67	79.27	83.13	85.27	82.87
Thyroid	85.46	85.55	85.61	78.07	86.21
Wisconsin	83.95	83.90	86.39	86.48	86.00

Problema	J48	PART	Naive Bayes	SMO (SVM)	RBFNN
Bal. Scale	0.93	1.35	0.24	0.30	0.94
Bupa	2.08	1.48	0.77	0.86	2.00
Diabetes	1.53	1.19	0.49	0.24	0.83
Glass	1.56	2.51	1.67	1.19	2.69
Iris	1.54	1.42	0.55	0.39	1.16
Thyroid	1.40	1.46	0.29	0.46	0.63
Wisconsin	0.63	1.00	0.05	0.10	0.31

La inclusión de ruido hace a APSC más competitivo en los dominios Balance Scale, Thyroid y Wisconsin. Este resultado se puede comparar con el mostrado en el Capítulo 5, Tabla 5.11, donde se podía observar que, con los conjuntos de datos originales, APSC empataba con otros algoritmos en dichos dominios.

Con respecto a ENPC_d, APSC mejora en la comparación, al pasar a obtener el mismo resultado en el dominio Balance Scale.

En suma, al incluir dominios con ruido APSC se ve especialmente beneficiado al comparar con los algoritmos de vecino más próximo, que son los

Tabla 6.3 Experimentación en presencia de un 10 % de ruido de clase. Comparación de APSC con algoritmos basados en el vecino más próximo (tabla superior) y resto de algoritmos (tabla inferior). El mejor resultado se marca en negrita sobre gris. Los símbolos indican el resultado del test de significación: "+" indica que APSC es mejor que el algoritmo indicado en la columna con $\alpha = 0.05$; "++" APSC es mejor con $\alpha = 0.01$; el signo negativo indica que el algoritmo de la columna es mejor que APSC.

Problema	APSC	IBK (K=1)	IBK (K=3)	LVQ	ENPC	ENPC$_d$
Balance Scale	**78.55**	++	++	++	++	=
Bupa	**62.61**	++	+	++	++	++
Diabetes	**70.01**	++	++	++	++	++
Glass	**71.66**	++	++	++	++	++
Iris	**85.35**	++	++	++	++	+
Thyroid	**85.27**	++	+	++	++	=
Wisconsin	**86.25**	++	+	+	+	=
Mejor	7	0	0	0	0	3

Problema	APSC	J48	PART	Naive Bayes	SMO (SVM)	RBFNN
Balance Scale	78.55	++	++	− −	−	+
Bupa	**62.61**	=	=	++	++	+
Diabetes	70.01	++	++	=	− −	+
Glass	**71.66**	++	++	++	++	++
Iris	**85.35**	++	++	+	=	+
Thyroid	**85.27**	=	=	=	++	=
Wisconsin	**86.25**	++	++	=	=	=
Mejor	5	2	2	3	3	2

más sensibles al ruido en líneas generales.

Para los algoritmos no basados en el vecino más próximo, si comparamos los resultados de la Tabla 6.3 con los resultados sin ruido (ver Capítulo 5, Tabla 5.11) se observa que los algoritmos conservan su posición relativa entre sí en casi todos los casos. Podemos observar también que J48 y PART mejo-

ran sus resultados en Thyroid, mientras que las RBFNN se ven perjudicadas, pasando a ser menos competitivas en Bupa e Iris.

APSC pasa a mejorar a las RBFNN en Balance-Scale, Bupa, Diabetes y Thyroid, dominios en los que previamente era peor o igual, lo cual indica un mejor comportamiento ante el tipo de ruido introducido.

En la Tabla 6.4 se muestran los resultados de APSC y de los algoritmos basados en el vecino más próximo. El dato que se muestra es la reducción del porcentaje de éxito en clasificación, sobre el conjunto de validación. La misma información se muestra de forma gráfica en la Figura 6.1, en la que mostramos en ordenadas la reducción en porcentaje de éxito, y en abcisas los dominios sobre los que se ha realizado la experimentación. En la Tabla 6.5 y la Figura 6.2 se muestra, en el mismo formato, la información correspondiente a los algoritmos no basados en el vecino más próximo.

En la Tabla 6.4 se observa que APSC tiene el mejor comportamiento medio, seguido de LVQ y ENPC$_d$. En media, la reducción en APSC es de un 7.52 %, menor que el porcentaje de ruido introducido. Por el contrario, se verifica que el algoritmo 1-NN (IBK-1) es muy sensible al ruido. Igualmente ENPC resulta muy sensible al ruido; con los datos detallados de la experimentación se ha verificado que esto es especialmente cierto en algunos casos en los que el algoritmo tiende a sobre entrenar, lo cual lo aproxima al comportamiento del algoritmo 1-NN.

En la Figura 6.1 se puede comprobar gráficamente cómo el comportamiento de APSC es el mejor o muy próximo al mejor en todos los dominios, siendo ENPC y 1-NN los algoritmos que se ven siempre más afectados por el ruido, con excepción del dominio Bupa, en el que los que se comportan peor son LVQ y ENPC$_d$.

En la Tabla 6.5 se observa que los algoritmos tienen un comportamiento más variable según cada dominio, dadas sus distintas características. En cuanto a los comportamientos medios, tanto en términos absolutos como relativos los mejores algoritmos son APSC, Naive Bayes y SMO.

En la Figura 6.2 se observa que el mejor algoritmo varía en cada dominio; pero, con la excepción de Thyroid y Bupa, APSC es el mejor o está próximo al mejor en muchos casos.

Por su parte, a pesar de su buen comportamiento medio, Naive Bayes tiene una importante variabilidad según el dominio, resultando especialmente sensible en el caso de Glass e Iris. SMO, sin embargo, parece ser bastante

Tabla 6.4 Reducción del porcentaje de éxito en validación cuando se introduce 10 % de ruido en las clases, para APSC y los algoritmos basados en el vecino más próximo

Problema	APSC	IBK (K=1)	IBK (K=3)	LVQ	ENPC	ENPC$_d$
Bal.Scale	7.09	13.20	8.38	8.41	24.73	7.44
Bupa	2.64	3.21	2.21	3.74	2.35	6.94
Diabetes	5.04	9.88	8.53	6.10	10.17	4.79
Glass	5.04	12.82	9.07	5.24	10.37	7.09
Iris	11.54	20.40	13.60	13.53	23.60	11.94
Thyroid	11.01	17.72	10.89	11.52	19.95	10.67
Wisconsin	10.26	15.70	11.35	10.62	11.32	11.19
Media	7.52	13.28	9.15	8.45	14.64	8.58

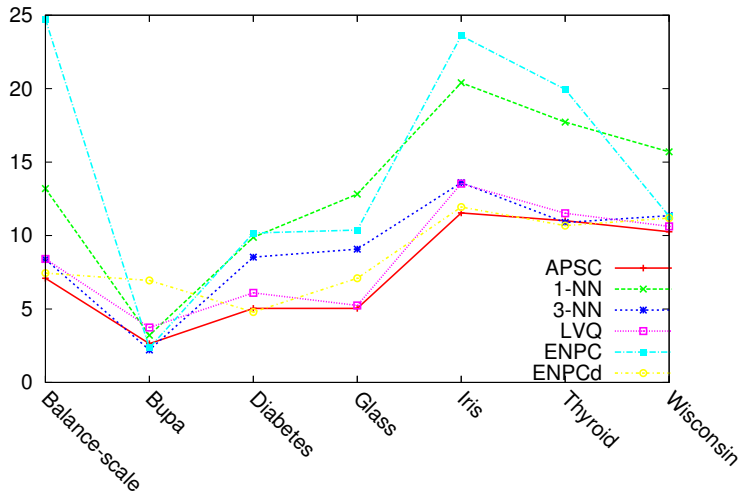

Figura 6.1 Gráfica de la influencia del ruido para los algoritmos basados en el vecino más cercano. En ordenadas, reducción del porcentaje de éxito en validación; en abcisas, dominios

homogéneo en su buena tolerancia al ruido.

Tabla 6.5 Reducción del porcentaje de éxito en validación cuando se introduce 10 % de ruido en las clases, para APSC y los algoritmos no basados en el vecino más próximo

Problema	APSC	J48	PART	N.Bayes	SMO	RBFNN
Bal.Scale	7.09	6.07	9.58	8.79	7.83	8.90
Bupa	2.64	4.18	1.62	0.54	0.65	4.78
Diabetes	5.04	7.48	6.46	5.75	4.48	5.80
Glass	5.04	10.19	12.28	2.69	8.20	9.24
Iris	11.54	14.06	14.93	12.20	11.00	13.13
Thyroid	11.01	6.60	8.37	11.14	11.67	10.20
Wisconsin	10.26	10.75	11.24	9.60	10.37	10.08
Media	7.51	8.48	9.21	7.24	7.74	8.88

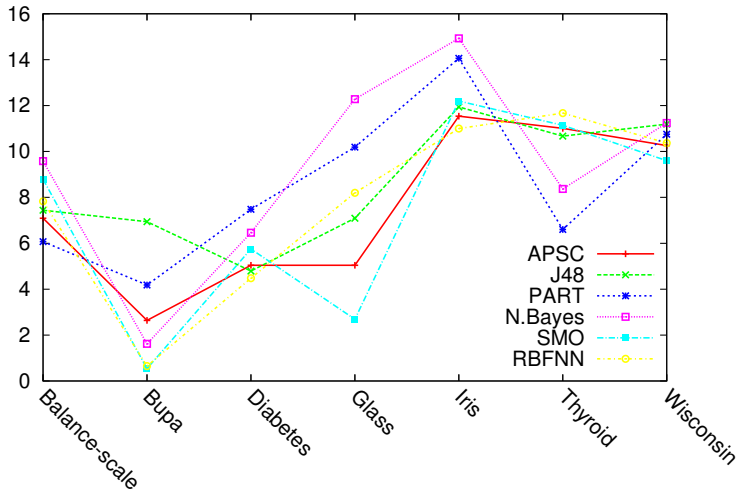

Figura 6.2 Gráfica de la influencia del ruido para los algoritmos no basados en el vecino más cercano. En ordenadas, reducción del porcentaje de éxito en validación; en abcisas, dominios

6.2 Discusión sobre la Experimentación en Presencia de Ruido

Los algoritmos de tipo vecino más próximo son especialmente sensibles al ruido en la clase porque cada patrón mal clasificado afecta al resultado

de la clasificación en toda la región de Voronoi que le corresponde.

En el caso de los algoritmos de reemplazo de prototipos, el patrón "erróneo" puede condicionar la posición de los prototipos, pero no la determina de forma exclusiva, sino que depende de otros patrones del conjunto de entrenamiento. De este modo, el efecto del ruido se ve reducido.

En LVQ, ENPC y ENPC$_d$, que son algoritmos en los que los prototipos se sitúan cerca del centro de los clusters de patrones, se verifica que el porcentaje de variación del resultado sigue el comportamiento esperado, mejorando de forma importante el resultado de 1-NN (ver Tabla 6.1).

En el caso de APSC, su resultado también mejora notablemente los resultados de 1-NN en validación, siendo el mejor algoritmo de los basados en el vecino más próximo, en términos medios, junto con ENPC$_d$ (ver Tabla 6.3).

Los resultados confirman que los algoritmos de reemplazo de prototipos basados en técnicas evolutivas (ENPC y APSC) tienen un buen comportamiento en presencia de ruido. Además, la forma en que APSC sitúa los prototipos (cerca de las fronteras de decisión) no parece hacerlo más sensible al ruido que ENPC$_d$ en estos dominios, pese a que este último sitúa los prototipos cerca de los centroides de los clusters.

Al comparar con otros algoritmos, basados en enfoques distintos, APSC sigue teniendo unas características de tolerancia al ruido comparables y resulta por tanto competitivo en este aspecto (ver Tabla 6.3).

7

PSC y APSC con Medida de Proximidad Local

Los algoritmos PSC y APSC ofrecen buenos resultados como métodos de reemplazo de prototipos para clasificación por vecino más próximo; sin embargo, es posible completarlos con mecanismos que aborden algunos de los inconvenientes compartidos por los clasificadores basados en este principio.

Como se ha mencionado en el Apartado 2.1.2, algunas de las circunstancias que afectan al rendimiento de los clasificadores basados en el vecino más próximo son:

- Elección de la medida de proximidad. La distancia Euclídea es una distancia geométrica que puede o ajustarse a los requisitos del problema. La representación de los atributos del problema como vectores de $\Re^d$ puede no ser la que mejor refleja las semejanzas que determinan la clasificación, o que la frontera de decisión "óptima" no se pueda representar mediante una superficie poliédrica. De igual manera, las distintas escalas de valores o unidades en que se midan los atributos requiere al menos considerar siempre algún mecanismo de escalado, lo cual es análogo a utilizar una medida de distancia Euclídea con una ponderación diferente (un coeficiente) para cada atributo.
- Todo clasificador K-NN es muy sensible a la presencia de atributos irrelevantes; es decir, información que aparece en los datos pero que no está correlacionada con la clase de los patrones. La función de distancia Euclídea incorpora los atributos independientemente de su relevancia, y por lo tanto puede determinar que patrones que están "conceptualmente" próximos estén matemáticamente lejanos al diferir en el valor que dan a un atributo irrelevante.

- Al crecer el número de atributos, crece la dimensión del espacio de patrones; se demuestra [Aha et al., 1991] que, a medida que crece dicho espacio, el número de patrones que requiere el algoritmo 1-NN crece exponencialmente. De nuevo ello reduce el comportamiento de los algoritmos K-NN en problemas con un número elevado de atributos.

Existen varias técnicas para abordar los problemas citados; la más básica es procesar el conjunto de atributos para determinar la relevancia de los mismo (Selección de Atributos) o incluso generar atributos nuevos garantizando que sean relevantes. También las técnicas de escalado de los atributos pueden contribuir a ello. Todo ello constituye el objeto de los filtros que pueden utilizarse como paso previo e independiente al uso de cualquier clasificador.

De forma más genérica, existe también la posibilidad de ajustar la medida de proximidad al problema que se desea resolver. La medida seleccionada puede o no tener las propiedades de una distancia matemática, y por ello utilizaremos de forma genérica la palabra "proximidad" o "similitud" para referirnos a ella.

Como se ha visto en el Apartado 2.1.6, la selección de una medida de proximidad para un problema y un clasificador se puede abordar desde varias perspectivas:

- Selección de una distancia a priori mediante escalado o análisis de los datos, u optimización de la distancia de forma adaptativa durante la ejecución del clasificador
- Métodos exactos (medidas estadísticas) o uso de un algoritmo de búsqueda (por ejemplo, evolutivo)
- Uso de una distancia global (igual para todos el espacio $\Re^d$) o local (diferente según la zona en la que se evalúe)

Se ha decidido comprobar en qué medida se pueden mejorar los resultados de PSC y APSC utilizando uno de estos enfoques, tal y como se describe en el Apartado 3.4.3. El algoritmo se puede caracterizar como una optimización "a posteriori" de la medida de proximidad **local** óptima para un conjunto de prototipos que clasifican un problema dado. Es decir, se determinan los prototipos $q_i \in P$ primero; luego, se determina una medida d_i para cada prototipo, de forma que la distancia entre cada prototipo y cada patrón x_j se calcula la medida particular del prototipo: $Proximidad(\mathbf{x}_j, \mathbf{q}_i) = d_i(\mathbf{x}_j, \mathbf{q}_i)$.

Una vez calculada la proximidad, se determina cuál es el prototipo más próximo al patrón y este es el que atribuye al patrón la clase.

Para ello se ha realizado un conjunto de experimentos nuevos, sobre los mismos conjuntos de datos utilizados en el Capítulo 5, tanto sobre el algoritmo PSC como el APSC. El resultado serán dos nuevas variantes de dichos algoritmos: PSC con optimización (PSC_d) y APSC con optimización ($APSC_d$).

Los dominios y parámetros de los experimentos fueron los ya utilizados en las pruebas anteriores para PSC y APSC (Capítulo 4) pero se realizó un conjunto nuevo de experimentos. Los resultados presentados en esta sección corresponden a dicha serie nueva de experimentos, por lo que los datos que se muestran para los algoritmos sin optimización no coinciden con exactitud con la anterior serie, debido al carácter estocástico de los algoritmos y la variación de la semilla aleatoria utilizada. Se ha comprobado que las diferencias entre los experimentos realizados para un mismo algoritmo, en el Apartado 5 y los realizados en los Apartados que siguen, no son significativas.

El algoritmo utilizado se describe en el Apartado 3.4.3. En todos los experimentos se ejecutaron 50 iteraciones del algoritmo de optimización, con una población de 20 individuos, y un valor del parámetro $\eta = 0.7$. Este parámetro es un parámetro estándar de estrategias evolutivas, y corresponde con la tasa de variación de la varianza que se utiliza en la mutación de los individuos. El valor seleccionado para η está dentro de los valores habituales en estos algoritmos.

7.1 Resultados

En la Tabla 7.1 se muestra el porcentaje de éxito en clasificación, medido sobre el conjunto de validación, tanto para PSC_d como para $APSC_d$. Se indica el porcentaje antes y después del proceso de optimización, con el fin de poder evaluar la mejora introducida por al algoritmos de optimización de proximidad.

La mayor mejora se obtiene en los dominios Bupa y Diabetes tanto para PSC_d como $APSC_d$. En PSC_d también se obtiene una cierta mejora en Balance Scale, y ninguna en Glass. En $APSC_d$ no se obtiene ninguna mejora ni para Balance Scale ni para Glass.

El resultado de $APSC_d$ sigue siendo igual o mejor que el de PSC_d, como

Tabla 7.1 Resultados de PSC_d y $APSC_d$ en validación. Se muestra el resultado antes y después del proceso de optimización, y, entre paréntesis, la mejora (en porcentaje) conseguida en la optimización de la medida de proximidad

	PSC	PSC_d		APSC	$APSC_d$	
Bal. Scale	82.08	83.08	(+1.00)	85.44	85.43	(-0.01)
Bupa	64.50	67.67	(+3.18)	65.04	67.46	(+2.41)
Diabetes	74.43	76.11	(+1.68)	74.41	75.74	(+1.33)
Glass	74.23	74.23	(0.00)	76.70	76.61	(-0.10)
Iris	97.36	98.05	(+0.70)	96.52	97.45	(+0.93)
New Thyroid	96.58	97.29	(+0.71)	96.31	96.79	(+0.47)
Wisconsin	96.19	96.76	(+0.57)	96.60	96.95	(+0.36)

ocurría al comparar APSC con PSC (sin optimización). Se han realizado comparaciones entre todos los algoritmos propuestos, y las tablas se incluyen en el Apartado 9.1. La conclusión es que en caso de existir diferencias entre PSC_d y $APSC_d$, en general no son estadísticamente significativas salvo en el caso del dominio Balance Scale. Por lo tanto, la inclusión del algoritmo de optimización tiende a igualar el resultado de PSC y APSC en los dominios Bupa, Diabetes, Iris, New Thyroid y Wisconsin.

En la Tabla 7.2 se muestra el resultado de la comparación del porcentaje de éxito en validación de PSC_d con el resto de los algoritmos usados como referencia; en la parte superior, figuran los algoritmos que usan la regla del vecino más próximo; en la parte inferior, se muestra el resto. La notación es la utilizada con anterioridad, y se indica en leyenda. Los porcentajes de éxito de todos los algoritmos se pueden consultar en las Tablas 4.4 4.5.

Al comparar PSC_d con el resto de los algoritmos basados en el vecino más próximo, los resultados muestran que mejora los resultados obtenidos por PSC (ver Tabla 5.7). PSC_d es el mejor algoritmo en solitario en Bupa, Diabetes e Iris. En Thyroid iguala el mejor resultado (que era el de 1-NN previamente), y en algunos otros casos se incrementa la significación estadística de la comparación con otros algoritmos. Es significativo que se ha mejorado la comparación con el algoritmo $ENPC_d$, que precisamente incluye un mecanismo de adaptación local de la medida de proximidad.

Consecuencia de lo anterior es que, en esta comparación con los algoritmos basados en el vecino más próximo, PSC_d es el mejor algoritmo en seis

de los siete problemas.

En la parte inferior de la Tabla 7.2 se pueden consultar los resultados de los tests de significación realizados entre el resultado de PSC_d y el de los algoritmos de referencia no basados en el vecino más próximo. Al comparar dichos resultados con los obtenidos por PSC (ver Tabla 5.7) podemos observar que se mejoran los resultados, pasando a ser el mejor algoritmo en solitario en Bupa e Iris. En particular el algoritmo mejora con respecto a dos algoritmos: con respecto a RBFNN, mejora la significación del resultado al comparar con este algoritmo en Bupa, Diabetes, Iris y Wisconsin; con respecto a GAssist, que conseguía igualar el resultado de PSC en cuatro de los dominios (Balance Scale, Bupa, Diabetes y Wisconsin), PSC_d pasa a obtener mejor resultado que GAssist en los siete dominios.

En esta comparación PSC_d pasa a ser el mejor algoritmo en seis de los siete dominios analizados, siendo Naive Bayes y SMO los algoritmos que lo seguirían en esta medida.

De forma análoga a la anterior, en la Tabla 7.3 se muestra el resultado de la comparación del porcentaje de éxito en validación de $APSC_d$ con el resto de los algoritmos; se utiliza el mismo formato y notación que anteriormente, de forma que los algoritmos que usan la regla del vecino más próximo figuran en la parte superior, y el resto en la inferior. Los porcentajes de éxito de todos los algoritmos se pueden consultar en las Tablas 4.4 4.5.

Los resultados de los tests realizado entre $APSC_d$ y los algoritmos que usan la regla del vecino más próximo, se pueden comparar con los realizados con anterioridad entre APSC y dichos algoritmos, que se muestran en la Tabla 5.11. Se puede observar que en Bupa y Diabetes $APSC_d$ pasa a ser el mejor algoritmo en solitario, y de nuevo se observa que se incrementa la diferencia en la comparación con $ENPC_d$, que pasa a ser competitivo únicamente en Balance Scale. Se confirma por lo tanto que la optimización de la medida de proximidad compensa en algunos dominios la ventaja que obtenía $ENPC_d$ con respecto a APSC gracias a su mecanismo de adaptación local de distancia.

Igual que PSC_d, $APSC_d$ es el mejor algoritmo en seis de los siete dominios.

En la parte inferior de la Tabla 7.3 se pueden consular los resultados de los tests estadísticos de comparación de $APSC_d$ con algoritmos no basados en el vecino más próximo. La comparación con los resultados de APSC sin optimización de distancias (ver Tabla 5.11)) ofrece resultados muy similares

a los que se obtienen para PSC_d: es decir, se pasa a ser el mejor algoritmo en solitario en Iris, y se mejora la significación de la comparación con algunos de los algoritmos considerados, especialmente RBFNN y GAssist.

7.2 Discusión de los Resultados de PSC_d y APSC_d

El algoritmo de optimización de la función de proximidad es sólo una de varias posibilidades que permitirían aplicar una deformación local a la forma de la frontera de decisión que determina cada prototipo.

Se puede consultar el Apartado 9.1 y la Tabla 9.8 para obtener una visión global de la comparación entre los resultados de los diversos algoritmos.

En líneas generales, podemos resumir las conclusiones obtenidas en el apartado de experimentación como sigue:

- El algoritmo de optimización de la función de proximidad permite mejorar el resultado de PSC y APSC en los dominios sencillos, definidos estos como los que requieren menos prototipos (Bupa, Diabetes, Iris, Thyroid y Wisconsin).

- Observamos que no hay mejoría alguna en los dominios que requieren muchos prototipos (Glass y Balance Scale con APSC). Nuestra hipótesis es que el algoritmo de optimización, tal y como está planteado, tiene dificultad para funcionar cuando crece la dimensión del problema. En estos casos, la dimensión de la solución es relativamente elevada. El caso más claro es Glass, que al tener 9 atributos, requiere 81 parámetros por cada matriz M; si consideramos que APSC conserva en la solución 55 prototipos de media (Tabla 5.9), la dimensión del individuo para estrategias evolutivas sería 4455. Se trata de un espacio de búsqueda demasiado grande para los parámetros utilizados, por lo que en estos casos la optimización de la función de proximidad no aporta nada a la solución. En el caso de Balance Scale, es capaz de aportar una mejora en PSC (del orden de 13 prototipos) pero no sobre APSC (que genera del orden de 63 prototipos).

- Se observa también que, en ocasiones, la inclusión de la optimización de la función de proximidad permite mejorar el porcentaje de éxito incluso por encima de la introducción de reproducción de partículas (Bupa, Diabetes y Thyroid). Es decir, PSC_d resulta mejor que APSC

sin optimización. Es decir, una medida de proximidad adecuada permite obtener buenos resultados sin incrementar el número de prototipos (ver Apartado 9.1, Tablas 9.2, 9.3 y 9.6).

- $APSC_d$ permite mejorar el resultado de APSC para algunos dominios (Bupa, Iris y Wisconsin), permitiendo distanciarse del resto de los algoritmos, especialmente RBFNN y SVM, que son los que competían de forma más directa con APSC (ver Apartado 9.1, Tablas 9.2, 9.5 y 9.7).

- Se observó con anterioridad que APSC tiene dificultad para obtener mejor resultado que otros algoritmos en el dominio Wisconsin, y se achacó a la mayor dimensión del espacio de búsqueda. Observamos que la utilización de una medida de proximidad distinta de la distancia Euclídea en $APSC_d$ mejora el comportamiento del algoritmo en este dominio (ver Apartado 9.1, Tabla 9.7). La diferente ponderación de los atributos contribuye por lo tanto a reducir el efecto de la dimensión en este dominio.

Podemos deducir que existe un cierto compromiso entre el beneficio de realizar la optimización de una medida de proximidad local y el de activar la generación de prototipos en APSC; sin embargo, cuando APSC es necesario para alcanzar resultados realmente competitivos (Balance Scale), el proceso de optimización no es suficiente como alternativa.

Existen formas de evitar que el algoritmo de optimización sea efectivo en casos en los que haya un número elevado de prototipos; para ello puede usarse un proceso evolutivo que optimice la matriz de la función de proximidad sólo para algunos prototipos. La caracterización de cuáles son susceptibles de mejorar puede hacerse mediante una medida de sus características de clasificación, o bien puede determinarse también por un método evolutivo. Cabe la posibilidad de diseñar un sistema coevolutivo que realice esta tarea, como el modelo coevolutivo propuesto por [Gagnec and Parizeau, 2007].

En cuanto a la forma de ejecutar el algoritmo de optimización, se ha optado por utilizarlo para refinar la solución final. Planteamos inicialmente la posibilidad de ejecutarlo durante la ejecución del algoritmo (A)PSC; la experimentación preliminar realizada en este caso determinó que, tal y como está definida, la optimización no mejoraba el comportamiento de APSC sino que lo empeoraba. Se observaron los siguientes efectos al incluir la optimización de proximidad en el curso de la ejecución del algoritmo APSC:

- En líneas generales se detectó que dicha forma de ejecución generaba soluciones con un menor número de prototipos; es decir, cada vez que se realizaba una fase de optimización de la función de proximidad, algunos prototipos que formaban parte de la solución quedaban descartados. Esto llevaba al estancamiento del enjambre en peores soluciones que las obtenidas por el algoritmo sin optimización de proximidad.

- Por otro lado, el cálculo de la proximidad entre patrones y partículas con la medida optimizada es más costoso que el cálculo con la distancia Euclídea; ello incrementaría notablemente el coste computacional de una versión de PSC que usara la medida de proximidad durante toda la ejecución del algoritmo.

- En tercer lugar, si la proximidad entre partículas y prototipos no se mide con la distancia Euclídea, cabría también analizar en qué medida deben resultar afectadas las relaciones de vecindad entre partículas; la distancia Euclídea se utiliza en otras partes del algoritmo, notablemente en la selección de los vecinos que afectan el movimiento de las partículas. Debería usarse también una medida diferente para determinar la partícula "más cercana" que ejerce influencia sobre una partícula dada.

Tabla 7.2 Comparación de PSC_d con algoritmos basados en el vecino más próximo (tabla superior) y resto de algoritmos (tabla inferior). El mejor resultado se marca en negrita sobre gris. Los símbolos indican el resultado del test de significación: "+" indica que PSC es mejor que el algoritmo indicado en la columna con $\alpha = 0.05$; "++" PSC es mejor con $\alpha = 0.01$; el signo negativo indica que el algoritmo de la columna es mejor que PSC.

	PSC_d	IBK-1	IBK-3	LVQ	ENPC	$ENPC_d$
Balance Scale	83.08	=	++	– –	=	– –
Bupa	**67.67**	++	++	++	++	++
Diabetes	**76.11**	++	++	++	++	++
Glass	**74.23**	++	++	++	++	++
Iris	**98.05**	++	++	++	++	++
Thyroid	**97.29**	=	++	++	++	++
Wisconsin	**96.76**	++	=	++	=	+
Mejor	6	1	1	0	1	1

	PSC_d	J48	PART	N.Bayes	SMO	RBFNN	GAssist	FRBCS
Bal. Scale	83.08	++	=	– –	– –	– –	+	
Bupa	**67.67**	+	++	++	++	+	++	
Diabetes	**76.11**	+	++	=	=	++	++	++
Glass	**74.23**	=	=	++	++	++	++	++
Iris	**98.05**	++	++	++	++	++	++	++
Thyroid	**97.29**	++	++	=	++	=	++	++
Wisconsin	**96.76**	++	++	++	=	++	+	++
Mejor	6	1	1	3	2	1	0	0

Tabla 7.3 Comparación de $APSC_d$ con algoritmos basados en el vecino más próximo (tabla superior) y resto de algoritmos (tabla inferior). El mejor resultado se marca en negrita sobre gris. Los símbolos indican el resultado del test de significación: "+" indica que PSC es mejor que el algoritmo indicado en la columna con $\alpha = 0.05$; "++" PSC es mejor con $\alpha = 0.01$; el signo negativo indica que el algoritmo de la columna es mejor que PSC.

	$APSC_d$	IBK-1	IBK-3	LVQ	ENPC	$ENPC_d$
Balance Scale	85.43	++	++	=	++	− −
Bupa	**67.46**	++	++	++	++	++
Diabetes	**75.74**	++	++	++	++	++
Glass	**76.61**	++	++	++	++	++
Iris	**97.45**	++	++	++	++	++
Thyroid	**96.79**	=	++	++	++	++
Wisconsin	**96.95**	++	=	++	=	++
Mejor	6	1	1	0	1	1

	$APSC_d$	J48	PART	N.Bayes	SMO	RBFNN	GAssist	FRBCS
Bal. Scale	85.43	++	++	− −	− −	=	++	++
Bupa	**67.46**	=	+	++	++	+	++	++
Diabetes	75.74	=	++	=	−	+	++	++
Glass	**76.61**	++	++	++	++	++	++	++
Iris	**97.45**	++	++	++	+	++	++	++
Thyroid	**96.79**	++	++	=	++	=	++	++
Wisconsin	**96.95**	++	++	++	=	++	++	++
Mejor	5	1	0	2	2	1	0	0

8

Conclusiones y Líneas Futuras de Investigación

En este capítulo destacamos las conclusiones que se obtienen de la experimentación realizada, así como de la comparación de las diversas variantes del algoritmo entre sí y con otros algoritmos.

También reflejamos las consecuencias que podemos extraer del trabajo realizado en términos generales. En algunas ocasiones nos referimos de forma conjunta a las dos versiones PSC y APSC del algoritmo propuesto, para lo cual utilizamos el término (A)PSC.

8.1 Conclusiones Teóricas

El Clasificador mediante Enjambre de Prototipos (A)PSC, representa una nueva aproximación para el problema de reemplazo de prototipos para aplicaciones de clasificación por prototipo más próximo. Se basa en un principio de aprendizaje inductivo supervisado y se abordada con un algoritmo de tipo metaheurístico, basado en población, y estocástico. El modelo computacional en el que se inspiran es el del algoritmo de Optimización mediante Enjambre de Partículas (PSO), si bien en nuestro caso la inspiración biológica no sería tanto un modelo de bandada que busca un lugar óptimo o una formación ideal, sino un modelo de varias especies de individuos independientes que deben distribuirse para localizar y compartir los recursos del entorno.

En algoritmos desarrollados en el campo de la reducción de prototipos (sea por selección o por reemplazo), es habitual centrar la medida de su utilidad en la capacidad de representación de un conjunto de patrones da-

do. Correspondería en nuestro caso con la exactitud en la clasificación del conjunto de entrenamiento. Sin embargo, al orientar (A)PSC a la aplicación en el campo de la Clasificación, hemos considerado que el criterio de evaluación más interesante lo constituye la capacidad de generalización, es decir, la calidad de la predicción que se realiza sobre patrones desconocidos.

Los algoritmos (A)PSC presentan algunas diferencias fundamentales con la mayoría de los algoritmos de reemplazo de prototipos más habituales, que detallamos a continuación.

Por una parte, los algoritmos (A)PSC introducen un sesgo específico en la búsqueda de dichos prototipos, ya que deliberadamente la enfocan hacia las fronteras de decisión. Esta elección se considera que puede mejorar la calidad del clasificador resultante en términos de capacidad de generalización, ya que los algoritmos que sitúan los prototipos en el centro de los agrupamientos de patrones requieren mecanismos adicionales para representar fronteras de decisión complejas.

Sabemos que existen métodos analíticos para determinar las fronteras de decisión que corresponden a un conjunto de entrenamiento (ver 2.1.3). Sin embargo, los algoritmos para dicho cálculo tienen una complejidad elevada, que los hace impracticables en el caso de espacios de dimensión elevada. En el contexto de la clasificación, la dimensión del dominio (número de atributos) hace inabordable el problema desde esta perspectiva, en una gran mayoría de las aplicaciones. Por este motivo resulta necesaria una aproximación al problema de tipo metaheurístico.

En (A)PSC se parte de un modelo de enjambre de individuos (partículas que representan un prototipo cada una), y, de forma similar a la que ocurre en un algoritmo evolutivo, se determina que las interacciones que dan lugar al movimiento de los individuos se establecerán únicamente entre dichos individuos. La interacción con el entorno (datos de entrenamiento) se realiza únicamente a través de la función de fitness que se pretende optimizar. En ello se diferencia de métodos de la familia de LVQ, que utilizan una heurística de posicionamiento en la que intervienen tanto los prototipos como los patrones. En estos métodos existe una función implícita que la heurística intenta optimizar, pero sólo en algunos dicha función forma parte de la definición del algoritmo.

En el algoritmo (A)PSC, la calidad de los prototipos viene dada por una función local que se puede variar: cabría usar una función que fijara criterios de calidad diferentes , o incluso utilizar un enfoque multicriterio

(optimización multiobjetivo).

En el estudio realizado con (A)PSC se ha definido una función de **fitness local jerárquica**, que proporciona buenos resultados en los problemas analizados. Al ser local, se trata de una función **multimodal**, para la que las partículas buscan soluciones diferentes en las diversas regiones del espacio. Para que dicha optimización local simultánea sea posible, las partículas están conectadas entre sí mediante un **vecindario dinámico geográfico**, es decir, definido por su situación en el espacio de búsqueda, como se requiere en las versiones multimodales de PSO. De esta forma, el enjambre se divide y recombina en subenjambres, de forma dinámica, en función de la situación de las partículas del enjambre en cada iteración del mismo.

Los Enjambres de Prototipos (A)PSC utilizan un codificación de Michigan (una partícula codifica un solo prototipo) para representar las soluciones. Ello tiene interés, respecto a aproximaciones tradicionales (Pittsburgh), ya que reduce el espacio de búsqueda. Sin embargo, tiene como consecuencia que el valor de la función de fitness local de una partícula cambia como consecuencia de la posición de las partículas vecinas. Por lo tanto, el enjambre busca optimizar una **función multimodal dinámica**. En los casos de estudio considerados, el conjunto de patrones que se utiliza para entrenamiento no varía con el tiempo; sin embargo, esta capacidad de optimización dinámica puede hacer al algoritmo aplicable en problemas de clasificación incremental, en los que el conjunto de patrones de entrenamiento varía (normalmente se incrementa) con el tiempo.

Por último, se ha verificado la utilidad de añadir al sistema de reemplazo de prototipos un mecanismo de adaptación de la medida de proximidad. El estudio realizado sólo ha explorado una versión básica de dicho mecanismo, por lo que existe campo para mejoras futuras.

8.2 Conclusiones del Estudio Experimental

En este trabajo se han realizado estudios experimentales con las diferentes variantes del algoritmo en un conjunto de dominios artificiales y reales. Para todos los casos, se ha realizado una comparación basada en tests de significación estadística que permiten evaluar la certeza en las comparaciones de los resultados. Las conclusiones que detallamos a continuación se basan en dichas comparaciones y no en el estudio de casos aislados.

8.2.1. Enfoque de Pittsburgh y Enfoque de Michigan

La primera comparación que queremos realizar concierne a la utilización de la versión de Pittsburgh. Si bien este enfoque se utiliza en otros trabajos [van der Merwe and Engelbrecht, 2003] [Falco et al., 2006], creemos que dichos estudios se ven limitados por la creación de un único prototipo por clase. En un algoritmo que intentara utilizar más prototipos, la dimensión del espacio de búsqueda crecería y se harían notorios los problemas de escalabilidad del enfoque de Pittsburgh.

También es relevante el hecho de que la versión de Pittsburgh no permite incorporar la adaptación del número de partículas (APSC) de forma natural, puesto que en un algoritmo PSO estándar la dimensión de la partícula es fija. Se ha comprobado que la estimación inicial del número de prototipos del problema es importante, y que la implementación de un mecanismo simple como el de APSC puede permitir al algoritmo corregir una estimación inicial equivocada.

El algoritmo PSC es capaz de generar conjuntos de prototipos bastante reducidos que representan correctamente el dominio, en el sentido de que obtiene porcentajes de éxito similares o mejores a los que obtiene el algoritmo básico de vecino más próximo (1-NN). Ello se consigue además con un coste computacional reducido al compararlo con la versión de Pittsburgh, cuando éste se mide en número de evaluaciones de la función de proximidad.

8.2.2. Efecto de la adaptación de la población

La versión de PSC con población adaptativa (APSC) se ha introducido con el fin de reducir la restricción que supone la determinación a priori del número adecuado de prototipos conveniente para representar la solución a un problema dado. Ello se lleva cabo mediante el mecanismo de reproducción de partículas propuesto en 3.4.1.

Según los resultados obtenidos, este mecanismo nunca deteriora el comportamiento del algoritmo en términos de porcentaje de éxito, aunque sí en términos de coste computacional. Ahora bien, se ha demostrado que existen dominios para los que la reproducción de partículas es necesaria si se estima de forma incorrecta la composición de la población inicial del enjambre. En uno de los dominios (Balance Scale), el uso de APSC consigue que el algoritmo sea competitivo, a pesar de no llegar a ser el mejor algoritmo de los considerados. En otros dominios (Bupa, Diabetes, Glass, Iris y New

Thyroid), APSC mejora PSC haciendo más significativa la comparación estadística con el resultado de otros algoritmos.

Nuestra conclusión es que un mecanismo de reproducción de partículas es de gran importancia para la versatilidad del algoritmo y su menor sensibilidad al parámetro $|P_c|$ (número de prototipos por clase). Sin este mecanismo, se comporta como un algoritmo de reemplazo clásico en el que debe estimarse por anticipado, siendo sensible al mismo como se comprobó en el Apartado 5.1.1.

Por otro lado, al realizar experimentación no recogida en este trabajo, referida al caso límite (inicialización con una partícula), el algoritmo de reproducción genera una población de partículas insuficiente para ofrecer un resultado tan competitivo. Creemos que la razón por la que no se llega al mismo resultado es la necesidad de una inicialización con un cierto grado de diversidad. Dicha hipótesis debería ser confirmada en futuros estudios. Por el momento, para obtener resultados satisfactorios es preciso realizar una cierta estimación inicial del número de partículas/prototipos necesario ($|P_c|$), y distribuir dicha población inicial de forma aleatoria de forma que se pueda cubrir una parte sustancial del espacio de búsqueda.

8.2.3. Efecto de la optimización local de medida de proximidad

La inclusión del algoritmo de optimización de la función de proximidad debe considerarse de forma positiva en los casos en que el número de prototipos de la solución es suficientemente bajo. En éstos, el proceso de optimización mejora el rendimiento de PSC y APSC al compararlo con otros algoritmos.

Sin embargo, aparece la necesidad de un compromiso entre la utilización de esta forma de optimización y la inclusión del mecanismo de reproducción de partículas; a mayor número de prototipos en la solución, peor funciona el algoritmo de optimización.

El algoritmo de optimización de la función de proximidad propuesto permite que mejore de forma importante la comparación entre los algoritmos PSC y APSC con algoritmos no basados en la regla del vecino más próximo. Ello indica que este mecanismo permite reproducir características de las fronteras de decisión generadas por dichos algoritmos mediante la variación de la función de similitud que cada prototipo utiliza de forma local.

Por otro lado, el coste computacional del enfoque utilizado no es despreciable, de forma que su inclusión debería considerarse en función de los requisitos que puedan existir en este aspecto.

8.2.4. Características de PSC y APSC comparados con otros algoritmos

La experimentación realizada ha permitido comparar (A)PSC con un conjunto amplio de algoritmos, que utilizan diversos paradigmas de aprendizaje (ver Apartado 4.1).

Hemos observado que APSC es capaz de conservar y mejorar el resultado de los algoritmos basados en el vecino más próximo en todos los dominios. La excepción es ENPC$_d$, que puede dar mejores resultados en dominios que requieren un número elevado de prototipos con respecto a los usados en la inicialización.

La ventaja de APSC con respecto a ENPC$_d$ en los dominios sencillos se ve incrementada al añadir el mecanismo de optimización de la función de proximidad (APSC$_d$). Hemos de recordar que ENPC$_d$ utiliza un mecanismo de ponderación local de atributos para ajustar, en cada iteración, la función de distancia del prototipo a los patrones. En el caso de ENPC$_d$ el cálculo de dicha función se realiza de forma analítica, no evolutiva. Ello permite realizar dicho ajuste como parte de la evolución del algoritmo, solución que por el momento hemos descartado para PSC.

Se ha observado también que APSC suele obtener resultados similares a los que obtienen las RFBNN en los dominios considerados, si bien es precisamente frente a RFBNN donde se obtiene mayor diferencia al incluir la optimización de la función de proximidad en APSC$_d$.

Por otra parte, el análisis de tolerancia frente al ruido (ver Capítulo 6) apunta a que (A)PSC tiene unas buenas características en este sentido. Este comportamiento es común a los algoritmos de tipo evolutivo, ya que realizan pocas suposiciones sobre la estructura de los datos que aprenden. En el caso de (A)PSC, se ha verificado que el hecho de que no sitúe los prototipos en el centro de los agrupamientos de patrones no resulta perjudicado frente a algoritmos basados en la regla del vecino más próximo que sí lo hacen de ese modo (como ENPC).

8.2.5. Caracterización en función de los dominios

La caracterización de los problemas que resultan más adecuados para su resolución mediante un algoritmo concreto es una labor compleja. Sin embargo, la experimentación realizada permite avanzar hipótesis que ayudan a caracterizar (A)PSC en función de los dominios en los que su utilización sería más recomendable.

Por una parte, ofrece buenos resultados en los dominios Bupa, Iris y Thyroid. Estos son los que tienen el número más reducido de atributos para un número pequeño de clases. Por lo tanto, podemos suponer que el rendimiento de (A)PSC es mejor en dominios con menor número de atributos. Este efecto es de esperar si consideramos que todos los algoritmos basados en la regla del vecino más próximo son especialmente sensibles a la dimensión de dicho espacio de atributos.

En segundo lugar, destacan los resultados en el dominio Glass. La diferencia de este dominio con el resto es, por un lado, la existencia de un número elevado de clases con una distribución no equilibrada; y, por otro, el número relativamente alto de atributos. Nuestra hipótesis es que estas características perjudican especialmente el comportamiento de otros algoritmos. En algún caso, como es SMO, el algoritmo de referencia está desarrollado para problemas de dos clases y sólo se extiende a multiclase mediante técnicas adicionales. Creemos que APSC resulta especialmente adecuado para realizar la generación simultánea de prototipos de varias clases.

Por último, observamos que en los dominios Diabetes, Bupa y Wisconsin, la utilización de la versión con optimización de la medida de proximidad en APSC_d mejora los resultados de APSC. Por lo tanto, podemos suponer que en estos dominios la distancia Euclídea, aplicada sobre todos los atributos, no es la medida óptima de proximidad. En el caso del dominio Wisconsin, también la optimización de la medida de proximidad resulta útil para distanciar (A)PSC_d de los algoritmos de referencia.

8.3 Aportación a PSO

El esquema básico de algoritmo de PSO con enfoque de Michigan puede aplicarse en campos diferentes de la resolución de problemas de clasificación. En términos generales, podemos sugerir la aplicación de la perspectiva de Michigan para todo problema en el que la codificación de los individuos

debiera considerar la misma solución a todas las permutaciones de una serie de soluciones parciales.

En este sentido, cabría pensar en el desarrollo de una generalización del Enjambre de Prototipos a un marco de Enjambre de Partículas Generalizado para resolución de problemas de estas características.

Algunos ejemplos de aplicación que podrían especificarse en términos de enjambres con enfoque Michigan son los siguientes:

- Clasificación mediante reglas de inducción para dominios con atributos discretos. En este caso, cada partícula representaría una regla de inducción. Sería interesante comparar el comportamiento de un Enjambre de Partículas estilo Michigan para codificar reglas con el comportamiento de los Learning Classifier Systems (ver [Sigaud and Wilson, 2007]).

  Para abordar este problema convendría desarrollar una versión discreta de APSC; una posibilidad sería definir una codificación en la que la posición de la partícula estuviera constituida por un conjunto de longitud variable de cláusulas (predicados), que constituyeran la parte izquierda de una regla. El vecindario puede determinarse en función de las superposiciones entre las diversas reglas, en términos del número de patrones clasificados por cada una. Sin llegar a este extremo de desarrollo, se han realizado trabajos preliminares en este sentido, utilizando una versión binaria del Enjambre de Prototipos, en [Cervantes et al., 2005a] y [Cervantes et al., 2005b].

- Resolución del problema de búsqueda de Códigos de Corrección de Errores (Error-Correcting Codes); para ello cada partícula representaría un código y habría de definir un Enjambre Binario capaz de situar las partículas de forma que representasen cadenas de bits con máxima dispersión en el espacio disponible.

- Resolución del problema Radio Network Design (RND). El problema consiste en la distribución de un conjunto de emisores de ciertas características de cobertura, de forma que atienda las necesidades de comunicación sobre una superficie dada. El problema es bidimensional por naturaleza, pero de tipo combinatorio y no continuo, puesto que el conjunto de localizaciones posibles para las antenas (edificios, puntos elevados, etc.) es limitado. Por otro lado, la complejidad del mismo puede incrementarse desde la consideración de un conjunto de emisores idénticos y de características homogéneas, hasta un problema en

que cada emisor tiene características y costes diferentes. Cada partícula del enjambre representaría en este caso un emisor, y el espacio de su movimiento sería discreto.

- Resolución de problemas de optimización multiobjetivo. Los algoritmos PSO para optimización multiobjetivo ya aplican un enfoque Michigan, en el sentido de que cada solución encontrada es simplemente un punto del frente de Pareto, mientras que la solución al problema es un conjunto de soluciones. Cada partícula en estos algoritmos representa una sola solución. El valor de fitness de cada partícula depende básicamente de si se trata o no de una solución no dominada.

  La aportación de PSC consistiría en definir de forma explícita el vecindario y la función local que se optimiza, que en general no depende sólo de la posición, sino de la posición del resto de las partículas que forman parte del enjambre. En el caso de las soluciones no dominadas, puede existir un criterio de evaluación que dependa de su situación en una región más o menos poblada del frente de Pareto; en el caso de las partículas cuya posición es dominada por otras, cabría pensar que se desea optimizar su proximidad al punto más próximo conocido del frente de Pareto.

8.4 Líneas de Trabajo Futuras

En el Apartado 8.3 proponemos algunos trabajos encaminados a generalizar los principios que se han incluido en los algoritmos propuestos.

Por otro lado, existe la posibildad de mejorar los algoritmos de Enjambre de Prototipos algunos de los cuales apuntamos en este Apartado.

En el aspecto de la definición de los vecindarios, (A)PSC aplica una regla que tiende a atraer entre sí las partículas de distinta clase que son vecinos sobre el grafo de Voronoi, y por la frontera entre sus regiones de Voronoi forma parte de la frontera de decisión entre clases distintas; también procura que se establezca una fuerza de repulsión entre partículas de una misma clase que son vecinos sobre el grafo de Voronoi, puesto que tienden a competir por clasificar los mismos patrones

Esta definición de vecindario no se genera de forma explícita. Si bien no es deseable utilizar el algoritmo cálculo del grafo de Voronoi, existen algoritmos como algunas versiones de LVQ, SOM o el Growing Neural Gas Net-

work (GNGN, [Fritzke, 1995]), en los que se identifican de forma explícita los prototipos que son candidatos a tener dicha relación de vecindad. Para ello, para cada patrón se determina cuál es el prototipo más próximo, y además el *segundo* prototipo en orden de proximidad. Estos dos prototipos comparten una frontera que se sitúa en algún punto entre el patrón y el *segundo* vecino. En un enjambre que crease las relaciones de vecindario utilizando este mecanismo, no sólo se estarían aprendiendo las posiciones óptimas, sino la topología de los datos de entrenamiento. Proponemos utilizar dicha información para construir de forma dinámica los grafos de vecindad, tanto el utilizado para la atracción como el utilizado para la repulsión entre partículas.

Otro aspecto que resulta interesante es la posibilidad de aplicación de técnicas de optimización multiobjetivo en la medida de la calidad de los prototipos. Estas técnicas evaluarían varias medidas sobre la región local de cada prototipo (sensitividad, especificidad, precisión, etc.) al comparar la posición actual con la mejor posición de la partícula que lo representa. Es importante resaltar que las técnicas habituales para optimización multiobjetivo no son de aplicación directa, puesto que debe considerarse que el frente de Pareto para las funciones de fitness de cada partícula es dinámico. Por ello habría que introducir de forma más explícita ideas procedentes del campo emergente de Optimización Dinámica Multiobjetivo ([Farina et al., 2004]).

El carácter dinámico de la búsqueda no se ha explorado en profundidad. En sistemas de optimización dinámica cabe introducir un elemento predictivo [Hatzakis and Wallace, 2006], que guíe la búsqueda en función de la evolución histórica, pero no necesariamente (como ahora ocurre) hacia la mejor situación histórica. La memoria de cada partícula podría utilizarse de forma un poco más elaborada de forma que se incorpore a su movimiento.

En cuanto al mecanismo de adaptación de la población, como se indicó en el Apartado 3.4.1, inicialmente se realizó experimentación con un mecanismo de destrucción dinámica de partículas. El mecanismo fue descartado al verificar que con frecuencia reducía la diversidad del enjambre, y exigía además la inclusión de nuevos parámetros. Sin embargo, esperamos que la inclusión con un mecanismo de este tipo se pueda realizar con cierta facilidad.

La técnica de optimización de la medida de proximidad podría integrarse de formas alternativas en la evolución del algoritmo, como se describe en el Apartado 7.2. Sería de interés continuar el trabajo en las siguientes líneas:

- Uso de técnicas de optimización que permitan mejorar el resultado

en dominios con más prototipos. El algoritmo utilizado no obtiene buenas soluciones, probablemente debido a la dimensión del espacio de búsqueda, al optimizar la función de proximidad en ciertos dominios. Cabría intentar desarrollar algoritmos de optimización más eficientes, capaces de tratar estos problemas.

- Otra alternativa sería la incorporación de la evolución de las medidas locales de proximidad a la evolución del algoritmo (A)PSC. Esta incorporación podría ser de tipo coevolutivo [Gagnec and Parizeau, 2007], bien en línea con el algoritmo principal, incorporando la medida de proximidad en la información de la partícula y calculándola (como en [Fernandez and Isasi, 2008]) o haciéndola evolucionar mediante operadores específicos.

- Se podría combinar el algoritmo de optimización (a posteriori) o de coevolución (durante la ejecución del algoritmo PSO) con alguna heurística que determine qué prototipos requieren una medida de proximidad distinta de la que usan en el momento del cálculo. De este modo, se podría limitar el cálculo a algunos prototipos, reduciendo el coste computacional de cualquier solución adoptada. Igualmente, si sólo algunos prototipos utilizan una medida de proximidad no Euclídea, no se incrementaría el coste computacional de la clasificación durante el entrenamiento del enjambre.

9

Apéndices

9.1 Tablas Comparativas por Dominios

En esta sección incluimos los resultados obtenidos al realizar tests estadísticos de comparación entre las diversas versiones presentadas de PSC y los algoritmos utilizados en las secciones anteriores, dominio a dominio.

En estas tablas, los signos "+" y "++" indican que el algoritmo en la fila mejora el resultado del algoritmo en la columna. En el primer caso, el test de significación es válido con un nivel de certeza del 95 %, mientras que en el segundo, el nivel de certeza es del 99 %. Si el signo es negativo, el algoritmo de dicha fila tiene un resultado significativamente peor que el algoritmo de la columna.

Tabla 9.1 Balance Scale: Comparación estadística entre algoritmos

	PSC	APSC	PSC_d	$APSC_d$	IBK 1	IBK 3	LVQ	ENPC	$ENPC_d$
PSC		−−	=	−−	=	++	−−	=	−
APSC	++		++	=	++	++	=	++	−
PSC_d	=	−−		−−	=	++	−−	=	−
$APSC_d$	++	=	++		++	++	=	++	−

	J48	PART	Bayes	SMO	RBFN	GAssist	FRBCS
PSC	++	=	−	−	−	=	++
APSC	++	++	−	−	=	++	++
PSC_d	++	=	−	−	−	+	++
$APSC_d$	++	++	−	−	=	++	++

Tabla 9.2 Bupa: Comparación estadística entre algoritmos

	PSC	APSC	PSC_d	$APSC_d$	IBK 1	IBK 3	LVQ	ENPC	$ENPC_d$
PSC		=	−−	−−	++	+	++	++	=
APSC	=		−−	−−	++	++	++	++	=
PSC_d	++	++		=	++	++	++	++	++
$APSC_d$	++	++	=		++	++	++	++	++

	J48	PART	Bayes	SMO	RBFN	GAssist	FRBCS
PSC	=	=	++	++	=	=	++
APSC	=	=	++	++	=	+	++
PSC_d	+	++	++	++	+	++	++
$APSC_d$	=	+	++	++	+	++	++

Tabla 9.3 Diabetes: Comparación estadística entre algoritmos

	PSC	APSC	PSC_d	$APSC_d$	IBK 1	IBK 3	LVQ	ENPC	$ENPC_d$
PSC		–	– –	– –	++	=	=	++	=
APSC	+		– –	=	++	+	=	++	++
PSC_d	++	++		=	++	++	++	++	++
$APSC_d$	++	=	=		++	++	++	++	++

	J48	PART	Bayes	SMO	RBFN	GAssist	FRBCS
PSC	=	+	–	– –	=	=	++
APSC	=	++	=	– –	=	++	++
PSC_d	+	++	=	=	++	++	++
$APSC_d$	=	++	=	–	+	++	++

Tabla 9.4 Glass: Comparación estadística entre algoritmos

	PSC	APSC	PSC_d	$APSC_d$	IBK 1	IBK 3	LVQ	ENPC	$ENPC_d$
PSC		–	=	–	++	++	++	++	++
APSC	+		+	=	++	++	++	++	++
PSC_d	=	–		–	++	++	++	++	++
$APSC_d$	+	=	+		++	++	++	++	++

	J48	PART	Bayes	SMO	RBFN	GAssist	FRBCS
PSC	=	=	++	++	++	++	++
APSC	++	++	++	++	++	++	++
PSC_d	=	=	++	++	++	++	++
$APSC_d$	++	++	++	++	++	++	++

Tabla 9.5 Iris: Comparación estadística entre algoritmos

	PSC	APSC	PSC_d	$APSC_d$	IBK 1	IBK 3	LVQ	ENPC	$ENPC_d$
PSC		=	– –	=	+	+	=	+	+
APSC	=		– –	=	++	++	+	++	++
PSC_d	++	++		=	++	++	++	++	++
$APSC_d$	=	=	=		++	++	++	++	++

	J48	PART	Bayes	SMO	RBFN	GAssist	FRBCS
PSC	++	++	+	=	=	++	++
APSC	++	++	++	=	=	++	++
PSC_d	++	++	++	++	++	++	++
$APSC_d$	++	++	++	+	++	++	++

Tabla 9.6 New Thyroid: Comparación estadística entre algoritmos

	PSC	APSC	PSC_d	$APSC_d$	IBK 1	IBK 3	LVQ	ENPC	$ENPC_d$
PSC		=	– –	–	–	++	++	+	=
APSC	=		– –	=	=	++	++	++	+
PSC_d	++	++		=	=	++	++	++	++
$APSC_d$	+	=	=		=	++	++	++	++

	J48	PART	Bayes	SMO	RBFN	GAssist	FRBCS
PSC	++	+	=	++	=	++	++
APSC	++	++	=	++	=	++	++
PSC_d	++	++	=	++	=	++	++
$APSC_d$	++	++	=	++	=	++	++

Tabla 9.7 Wisconsin: Comparación estadística entre algoritmos

	PSC	APSC	PSC_d	$APSC_d$	IBK 1	IBK 3	LVQ	ENPC	$ENPC_d$
PSC		=	=	−−	++	=	+	=	=
APSC	=		=	−−	++	=	+	=	=
PSC_d	=	=		=	++	=	++	=	+
$APSC_d$	++	++	=		++	=	++	=	++

	J48	PART	Bayes	SMO	RBFN	GAssist	FRBCS
PSC	++	++	=	=	=	=	++
APSC	++	++	=	=	=	=	++
PSC_d	++	++	++	=	++	+	++
$APSC_d$	++	++	++	=	++	++	++

9.2 Resumen de Resultados

La Tabla 9.8 contiene un resumen de los resultados de la experimentación realizada con todos los algoritmos, sobre los dominios reales (datos UCI). Se muestra el porcentaje medio de éxito sobre los datos de validación, para diez experimentos con validación cruzada.

Tabla 9.8 Resumen de Resultados. Porcentaje de éxito sobre datos de validación, para todos los dominios y algoritmos

	Balance Scale	Bupa	Diabetes	Glass	Iris	New Thyroid	Wisconsin
PSC	82.59	64.76	74.25	74.23	96.70	95.90	96.50
APSC	85.64	65.25	75.05	76.70	96.89	96.28	96.51
PSC_d	83.08	67.67	76.11	74.23	98.05	97.29	96.76
$APSC_d$	85.43	67.45	75.74	76.61	97.45	96.79	96.95
IBK-1	82.42	62.22	70.40	69.07	95.40	96.93	95.37
IBK-3	80.26	62.48	73.89	69.05	95.20	94.31	96.54
LVQ	85.10	62.18	74.26	62.56	95.60	91.03	95.88
ENPC	82.66	59.41	68.42	68.48	95.33	94.81	96.86
$ENPC_d$	87.00	64.15	73.54	64.24	95.07	94.81	96.20
J48	77.82	66.01	74.68	73.61	94.73	92.62	94.59
PART	83.17	63.08	73.04	73.32	94.20	94.52	94.61
N. Bayes	90.53	55.63	75.70	46.23	95.33	96.79	96.07
SMO	87.62	57.95	76.93	57.81	96.27	89.30	96.74
RBFNN	86.25	65.09	74.38	65.45	96.00	96.41	96.08
GAssist	81.81	63.21	73.72	61.81	94.13	92.04	96.03
FRBCS	78.22	58.42	68.47	53.33	94.06	85.14	95.40
Promedio	83.73	62.81	73.66	66.67	95.65	94.06	96.07

Bibliografía

[Aha, 1992] Aha, D. W. (1992). Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms. *Int. J. Man-Mach. Stud.*, 36(2):267–287.

[Aha et al., 1991] Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance-based learning algorithms. *Mach. Learn.*, 6(1):37–66.

[Alcala-Fernández et al., 2008] Alcala-Fernández, J., Garcia, S., Berlanga, F., Fernández, A., Sanchez, L., del Jesus, M., and Herrera, F. (2008). Keel: A data mining software tool integrating genetic fuzzy systems. *Genetic and Evolving Systems, 2008. GEFS 2008. 3rd International Workshop on*, pages 83–88.

[Arya and Mount, 1993] Arya, S. and Mount, D. M. (1993). Approximate nearest neighbor queries in fixed dimensions. In *SODA '93: Proceedings of fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 271–280, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

[Asuncion and Newman, 2007] Asuncion, A. and Newman, D. (2007). UCI machine learning repository.

[Atkeson et al., 1997] Atkeson, C., Moore, A., and Schaal, S. (1997). Locally weighted learning. *AI Review*, 11:11–73.

[Aurenhammer, 1991] Aurenhammer, F. (1991). Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405.

[Bacardit and Garrell, 2007] Bacardit, J. and Garrell, J. (2007). *Bloat Control and Generalization Pressure Using the Minimum Description Length Principle for a Pittsburgh Approach Learning Classifier System*, volume 4399/2007 of *Lecture Notes in Computer Science*, pages 59–79. Springer Berlin / Heidelberg.

[Bacardit and i Guiu, 2003] Bacardit, J. and i Guiu, J. M. G. (2003). Evolving multiple discretizations with adaptive intervals for a pittsburgh rule-based learning classifier system. In Cantú-Paz, E., Foster, J. A., Deb, K., Davis, L., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Standish, R. K., Kendall, G., Wilson, S. W., Harman, M., Wegener, J., Dasgupta, D., Potter,

M. A., Schultz, A. C., Dowsland, K. A., Jonoska, N., and Miller, J. F., editors, *GECCO*, volume 2724 of *Lecture Notes in Computer Science*, pages 1818–1831. Springer.

[Bäck et al., 1991] Bäck, T., Rüdolph, G., and Schwefel, H. (1991). A survey of evolution strategies. In *Proceedings of 4th International Conference on Genetic Algorithms*, pages 2–9.

[Bäck and Schwefel., 1992] Bäck, T. and Schwefel., H. (1992). Evolutionary algorithms: Some very old strategies for optimization and adaptation. new computing techniques in physics research ii. In *Proc. Second Int'l Workshop Software Engineering, Artificial Intelligence, and Expert Systems for High Energy and Nuclear Physics*, pages 247–254.

[Baeck et al., 2000] Baeck, T., Fogel, D. B., and Michalewicz, Z. (2000). *Evolutionary Computation 1: Basic Algorithms and Operators (Evolutionary Computation)*. TF-TAYLOR.

[Baluja, 1994] Baluja, S. (1994). Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA.

[Bernadó-Mansilla and Garrell-Guiu, 2003] Bernadó-Mansilla, E. and Garrell-Guiu, J. M. (2003). Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. *Evol. Comput.*, 11(3):209–238.

[Bezdek and Kuncheva, 2000] Bezdek, J. C. and Kuncheva, L. (2000). Some notes on twenty one (21) nearest prototype classifiers. In *Proceedings of Joint IAPR International Workshops on Advances in Pattern Recognition*, pages 1–16, London, UK. Springer-Verlag.

[Blachnik and Duch, 2008] Blachnik, M. and Duch, W. (2008). *Prototype rules from SVM*, volume 80/2008 of *Studies in Computational Intelligence*, pages 163–182. Springer Berlin / Heidelberg.

[Blackwell, 2007] Blackwell, T. (2007). *Particle Swarm Optimization in Dynamic Environments*, pages 29–49. Studies in Computational Intelligence. Springer Berlin / Heidelberg.

[Blackwell and Bentley, 2002] Blackwell, T. and Bentley, P. J. (2002). Dynamic search with charged swarms. In *Proceedings of Genetic and Evolutionary Computation Conference 2002 (GECCO)*, pages 19–26.

[Boser et al., 1992] Boser, B. E., Guyon, I. M., and Vapnik, V.Ñ. (1992). A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of fifth annual workshop on Computational learning theory*, pages 144–152, New York, NY, USA. ACM.

[Bratton and Kennedy, 2007] Bratton, D. and Kennedy, J. (1-5 April 2007). Defining a standard for particle swarm optimization. *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*, pages 120–127.

[Brighton and Mellish, 2002] Brighton, H. and Mellish, C. (2002). Advances in instance selection for instance-based learning algorithms. *Data mining and knowledge discovery*, 6(2):153–172.

[Brits, 2002] Brits, R. (2002). Niching strategies for particle swarm optimization. Master's thesis, University of Pretoria, Pretoria.

[Brits et al., 2002] Brits, R., Engelbrecht, A. P., and Bergh, F. V. D. (2002). A niching particle swarm optimizer. In *In Proceedings of Conference on Simulated Evolution And Learning*, pages 692–696.

[Cagnoni and Valli, 1994] Cagnoni, S. and Valli, G. (1994). Oslvq: a training strategy for optimum-size learning vector quantization classifiers. In *Proceedings of IEEE World Congress on Computational Intelligence*, volume 2, pages 762–765 vol.2.

[Cano et al., 2003] Cano, J., Herrera, F., and Lozano, M. (2003). Using evolutionary algorithms as instance selection for data reduction in kdd: an experimental study. *IEEE Transactions on Evolutionary Computation*, 7(6):561–575.

[Cendrowska, 1987] Cendrowska, J. (1987). Prism: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4):349–370.

[Cervantes et al., 2005a] Cervantes, A., Isasi, P., and Galván, I. (2005a). Binary Particle Swarm Optimization in classification. *Neural Network World*, 15(3):229–241.

[Cervantes et al., 2005b] Cervantes, A., Isasi, P., and Galván, I. (2005b). A comparison between the Pittsburgh and Michigan approaches for the Binary PSO algorithm. In *Proceedings of IEEE Congress on Evolutionary Computation 2005 (CEC 2005)*, pages 290–297.

[Chang, 1974] Chang, C. (1974). Finding prototypes for nearest neighbour classification. *IEEE Transactions on Comput.*, C-23(11):1179 – 1184.

[Chen et al., 2005] Chen, J., Chen, H., and Ho, S. (2005). Design of nearest neighbor classifiers: multi-objective approach. *International Journal of Approximate Reasoning*, 40(1-2):3 – 22. Data Mining and Granular Computing.

[Chuang et al., 2008] Chuang, L., Chang, H., Tu, C., and Yang, C. (2008). Improved binary pso for feature selection using gene expression data. *Computational Biology and Chemistry*, 32(1):29 – 38.

[Cleary and Trigg, 1995] Cleary, J. G. and Trigg, L. E. (1995). K*: An instance-based learner using an entropic distance measure. In *In Proceedings of 12th International Conference on Machine Learning*, pages 108–114. Morgan Kaufmann.

[Clerc, 1999] Clerc, M. (1999). The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Proceedings of 1999 Congress on Evolutionary Computation, 1999. CEC 99.*, volume 3, pages –1957 Vol. 3.

[Clerc, 2000] Clerc, M. (2000). Discrete PSO: Travelling Salesman Problem. http://clerc.maurice.free.fr/pso/.

[Clerc, 2005] Clerc, M. (2005). Binary Particle Swarm Optimisers: toolbox, derivations, and mathematical insights. 42 pages.

[Clerc, 2006] Clerc, M. (2006). Stagnation Analysis in Particle Swarm Optimisation or What Happens When Nothing Happens. 17 pages.

[Clerc and Kennedy, 2002] Clerc, M. and Kennedy, J. (2002). The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Trans. Evolutionary Computation*, 6(1):58–73.

[Coello et al., 2004] Coello, C. A., Toscano, G., and Salazar, M. (2004). Handling multiple objectives with particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):256–279.

[Cohen, 1995] Cohen, W. W. (1995). Fast effective rule induction. In *In Proceedings of Twelfth International Conference on Machine Learning*, pages 115–123.

[Correa et al., 2006] Correa, E. S., Freitas, A. A., and Johnson, C. G. (2006). A new discrete particle swarm algorithm applied to attribute selection in a bioinformatics data set. In *GECCO '06: Proceedings of 8th annual conference on Genetic and evolutionary computation*, pages 35–42, New York, NY, USA. ACM.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). *Support-vector Networks*, volume 20, Number 3 /1995, pages 273–297. Springer Netherlands.

[Cost and Salzberg, 1993] Cost, R. S. and Salzberg, S. (1993). A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10:57–78.

[Cover and Hart, 1967] Cover, T. and Hart, P. E. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.

[de Almeida and Pozo, 2007] de Almeida, A. and Pozo, A. (2007). Multiple objective particle swarm for classification-rule discovery. In *Proceedings of IEEE Congress on Evolutionary Computation, 2007 (CEC 2007).*, pages 684–691.

[Devroye et al., 1996] Devroye, L., Gyorfi, L., and Lugosi, G. (1996). *A Probabilistic Theory of Pattern Recognition*. Stochastic modelling and applied probability. Springer, New York, NY, USA.

[D.Michie and D.J.Spiegelhalter, 1994] D.Michie and D.J.Spiegelhalter (1994). *Machine Learning, Neural and Statistical Classification*. Prentice Hall.

[Duch and Grudzinski, 2001] Duch, W. and Grudzinski, K. (2001). Prototype based rules-a new way to understand the data. In *Proceedings of International Joint Conference on Neural Networks, 2001 (IJCNN '01).*, volume 3, pages 1858–1863 vol.3.

[Eberhart and Shi, 2000] Eberhart, R. and Shi, Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of 2000 Congress on Evolutionary Computation, 2000.*, volume 1, pages 84–88 vol.1.

[Eberhart and Shi, 1998] Eberhart, R. C. and Shi, Y. (1998). Evolving artificial neural networks. In *Proceedings of International Conference on Neural Networks and Brain*, pages 5–13.

[Eshelman, 1990] Eshelman, L. J. (1990). The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In Rawlins, G. J. E., editor, *FOGA*, pages 265–283. Morgan Kaufmann.

[Esmin, 2007] Esmin, A. A. A. (2007). Generating fuzzy rules from examples using the particle swarm optimization algorithm. In *Proceedings of 7th International Conference on Hybrid Intelligent Systems (HIS 2007)*, pages 340–343. IEEE Computer Society.

[Esquivel and Coello, 2003] Esquivel, S. and Coello, C. (2003). On the use of particle swarm optimization with multimodal functions. In *Proceedings of IEEE Congress on Evolutionary Computation, 2003 (CEC '03).*, volume 2, pages 1130–1136 Vol.2.

[Falco et al., 2006] Falco, I. D., Cioppa, A. D., and Tarantino, E. (2006). *Evaluation of Particle Swarm Optimization Effectiveness in Classification*, volume 3849/2006 of *Lecture Notes in Computer Science*, pages 164–171. Springer Berlin / Heidelberg.

[Farina et al., 2004] Farina, M., Deb, K., and Amato, P. (2004). Dynamic multiobjective optimization problems: test cases, approximations, and applications. *IEEE Transactions on Evolutionary Computation*, 8(5):425–442.

[Fawcett, 2004] Fawcett, T. (2004). Roc graphs: Notes and practical considerations for researchers. Technical report, HP laboratories.

[Fernández and Isasi, 2004] Fernández, F. and Isasi, P. (2004). Evolutionary design of nearest prototype classifiers. *Journal of Heuristics*, 10(4):431–454.

[Fernandez and Isasi, 2008] Fernandez, F. and Isasi, P. (2008). Local feature weighting in nearest prototype classification. *IEEE Transactions on Neural Networks*, 19(1):40–53.

[Fieldsend and Singh, 2002] Fieldsend, J. and Singh, S. (2002). A multi-objective algorithm based upon particle swarm optimisation, an efficient data structure and turbulence. In *The 00 U.K. Workshop on Computational Intelligence*, volume 2723/2003, pages 34–44.

[Frank and Witten, 1998] Frank, E. and Witten, I. H. (1998). Generating accurate rule sets without global optimization. In *ICML '98: Proceedings*

of Fifteenth International Conference on Machine Learning, pages 144–151, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Fritzke, 1995] Fritzke, B. (1995). A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press.

[Gabriel and Sokal, 1969] Gabriel, R. K. and Sokal, R. R. (1969). A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18(3):259–278.

[Gagnec and Parizeau, 2007] Gagnec, C. and Parizeau, M. (2007). Coevolution of nearest neighbor classifiers. *International Journal of Pattern Recognition and Artificial Intelligence*, 21(5):921–946.

[Gates, 1972] Gates, G. (1972). The reduced nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 18(3):431–433.

[Gil-Pita and Yao, 2008] Gil-Pita, R. and Yao, X. (2008). Evolving edited k-nearest neighbor classifiers. *Int. J. Neural Syst.*, 18(6):459–467.

[Godfried T. Toussaint and Poulsen, 1984] Godfried T. Toussaint, B. K. B. and Poulsen, R. S. (1984). The application of voronoi diagrams to nonparametric decision rules. In *16th Symposium on Computer Science and Statistics*, pages 97–108.

[Hart, 1968] Hart, P. (1968). The condensed nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 14(3):515–516.

[Hatzakis and Wallace, 2006] Hatzakis, I. and Wallace, D. (2006). Dynamic multi-objective optimization with evolutionary algorithms: a forward-looking approach. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1201–1208, New York, NY, USA. ACM.

[Holden and Freitas, 2007] Holden, N. P. and Freitas, A. A. (2007). A hybrid pso/aco algorithm for classification. In *GECCO '07: Proceedings of 2007 GECCO conference companion on Genetic and evolutionary computation*, pages 2745–2750, New York, NY, USA. ACM.

[Holland, 1976] Holland, J. (1976). Adaptation. *Progress in theoretical biology*, IV:263–293.

[Hu and Eberhart, 2002] Hu, X. and Eberhart, R. (2002). Multiobjective optimization using dynamic neighborhood particle swarm optimisation. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, pages 1677–16.

[Hu et al., 2004] Hu, X., Shi, Y., and Eberhart, R. (2004). Recent advances in particle swarm. In *Proceedings of IEEE Congress on Evolutionary Computation 2004 (CEC 2004)*, pages 90–97.

[Huang and Dun, 2008] Huang, C. and Dun, J. (2008). A distributed pso-svm hybrid system with feature selection and parameter optimization. *Appl. Soft Comput.*, 8(4):1381–1391.

[Ishibuchi et al., 1999] Ishibuchi, H., Nakashima, T., and Murata, T. (1999). Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 29(5):601–618.

[Ishida et al., 2008] Ishida, C. Y., de Carvalho, A. B., Pozo, A. T. R., Goldbarg, E. F. G., and Goldbarg, M. C. (2008). Exploring multi-objective pso and grasp-pr for rule induction. In van Hemert, J. I. and Cotta, C., editors, *EvoCOP*, volume 4972 of *Lecture Notes in Computer Science*, pages 73–84. Springer.

[Iswandy and Koenig, 2008] Iswandy, K. and Koenig, A. (2008). Pso for fault-tolerant nearest neighbor classification employing reconfigurable, analog hardware implementation in low power intelligent sensor systems. In *Proceedings of Eighth International Conference on Hybrid Intelligent Systems, 2008 (HIS '08).*, pages 380–385.

[Jiang et al., 2007] Jiang, M., Luo, Y., and Yang, S. (2007). Stagnation analysis in particle swarm optimization. In *Proceedings of IEEE Swarm Intelligence Symposium, 2007 (SIS 2007).*, pages 92–99.

[Jin, 2006] Jin, Y. (2006). *Multi-Objective Machine Learning*, volume 16/2006 of *Studies in Computational Intelligence*. Springer Berlin / Heidelberg.

[Jin, 2007] Jin, Y. (2007). Pareto-based multi-objective machine learning. In *7th International Conference on Hybrid Intelligent Systems, 2007. HIS 2007.*, pages 2–2.

[Kennedy and Eberhart, 1995] Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks, 1995.*, volume 4, pages 1942–1948 vol.4.

[Kennedy and Eberhart, 1997] Kennedy, J. and Eberhart, R. (1997). A discrete binary version of the particle swarm algorithm. In *IEEE International Conference on Systems, Man, and Cybernetics, 1997. 'Computational Cybernetics and Simulation'., 1997*, volume 5, pages 4104–4108 vol.5.

[Kennedy et al., 2001] Kennedy, J., Eberhart, R., and Shi, Y. (2001). *Swarm intelligence*. Morgan Kaufmann Publishers, San Francisco.

[Kennedy and Mendes, 2002] Kennedy, J. and Mendes, R. (2002). Population structure and particle swarm performance. In *CEC '02: Proceedings of Evolutionary Computation on 2002. CEC '02. Proceedings of 2002 Congress*, pages 1671–1676, Washington, DC, USA. IEEE Computer Society.

[Kennedy and Spears, 1998] Kennedy, J. and Spears, W. (1998). Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator. In *IEEE International Conference on Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence.*, pages 78–83.

[Klein, 1989] Klein, R. (1989). *Concrete and Abstract Voronoi Diagrams*, volume 400 of *Lecture Notes in Computer Science*. Springer.

[Kohonen, 1995] Kohonen, T. (1995). *Self-Organizing Maps*. Springer Verlag, Berlin.

[Koza, 1992] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

[Kuncheva and Bezdek, 1998] Kuncheva, L. and Bezdek, J. (1998). Nearest prototype classification: clustering, genetic algorithms, or random search? *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 28(1):160–164.

[Li, 2004] Li, X. (2004). Adaptively choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization. In *in GECCO-2004*, pages 105–116. Springer-Verlag.

[Lim et al., 1991] Lim, G., Alder, M., and Hadingham, P. (1991). Adaptive quadratic neural nets. In *IEEE International Joint Conference on Neural Networks, 1991.*, pages 1943–1948 vol.3.

[Liu et al., 2006] Liu, Y., Qin, Z., Xu, Z., and He, X. (2006). *Classification Rule Mining Based on Particle Swarm Optimization*, volume 4062/2006 of *Lecture Notes in Computer Science*, pages 436–441. Springer Berlin / Heidelberg.

[Lloyd, 1982] Lloyd, S. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137.

[Marinakis et al., 2008] Marinakis, Y., Marinaki, M., and Dounias, G. (2008). Particle swarm optimization for pap-smear diagnosis. *Expert Systems with Applications*, 35(4):1645 – 1656.

[Matula and Sokal, 1980] Matula, D. W. and Sokal, R. R. (1980). Properties of gabriel graphs relevant to geographic variation research and the clustering of points in the plane. *Geographical Analysis*, 12:205–222.

[Melgani and Bazi, 2008] Melgani, F. and Bazi, Y. (2008). Classification of electrocardiogram signals with support vector machines and particle swarm optimization. *IEEE Transactions on Information Technology in Biomedicine*, 12(5):667–677.

[Meyer et al., 2003] Meyer, D., Leisch, F., and Hornik, K. (2003). The support vector machine under test. *Neurocomputing*, 55(1-2):169 – 186. Support Vector Machines.

[Nanni and Lumini, 2009] Nanni, L. and Lumini, A. (2009). Particle swarm optimization for prototype reduction. *Neurocomput.*, 72(4-6):1092–1097.

[Nickabadi et al., 2008] Nickabadi, A., Ebadzadeh, M., and Safabakhsh, R. (2008). Dnpso: A dynamic niching particle swarm optimizer for multimodal optimization. In *Proceedings of IEEE Congress on Evolutionary Computation, 2008 (CEC 2008). (IEEE World Congress on Computational Intelligence)*., pages 26–32.

[Omran et al., 2006] Omran, M. G., Salman, A. A., and Engelbrecht, A. P. (2006). Dynamic clustering using particle swarm optimization with application in image segmentation. *Pattern Anal. Appl.*, 8(4):332–344.

[O'Neill and Brabazon, 2008] O'Neill, M. and Brabazon, A. (2008). Self-organising swarm (soswarm). *Soft Comput.*, 12(11):1073–1080.

[Ozcan and Mohan, 1999] Ozcan, E. and Mohan, C. (1999). Particle swarm optimization: surfing the waves. In *Proceedings of 1999 Congress on Evolutionary Computation, 1999. CEC 99.*, volume 3, pages –1944 Vol. 3.

[Pan et al., 2008] Pan, Q., Fatih Tasgetiren, M., and Liang, Y. (2008). A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. *Computers and Operations Research*, 35(9):2807–2839.

[Parsopoulos and Vrahatis, 2001] Parsopoulos, K. E. and Vrahatis, M.Ñ. (2001). Modification of the particle swarm optimizer for locating all the global minima. In *Karny (Eds.), Artificial Neural Networks and Genetic Algorithms*, pages 324–327. Springer.

[Parsopoulos and Vrahatis, 2002] Parsopoulos, K. E. and Vrahatis, M.Ñ. (2002). Recent approaches to global optimization problems through particle swarm optimization. *Natural Computing*, 1:235–306.

[Passaro and Starita, 2008] Passaro, A. and Starita, A. (2008). Particle swarm optimization for multimodal functions: a clustering approach. *J. Artif. Evol. App.*, 8(2):1–15.

[Poli, 2008a] Poli, R. (2008a). Analysis of the publications on the applications of particle swarm optimisation. *J. Artif. Evol. App.*, 2008(1):1–10.

[Poli, 2008b] Poli, R. (2008b). Dynamics and stability of the sampling distribution of particle swarm optimisers via moment analysis. *J. Artif. Evol. App.*, 2008(2):1–10.

[Poli and Broomhead, 2007] Poli, R. and Broomhead, D. (2007). Exact analysis of the sampling distribution for the canonical particle swarm optimiser and its convergence during stagnation. In *GECCO '07: Proceedings of 9th annual conference on Genetic and evolutionary computation*, pages 134–141, New York, NY, USA. ACM.

[Powell, 1987] Powell, M. J. D. (1987). *Radial basis functions for multivariable interpolation: a review*, pages 143–167. Clarendon Press Institute of Mathematics and its Applications. Clarendon Press, New York, NY, USA.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Rameshkumar et al., 2005] Rameshkumar, K., Suresh, R., and Mohanasundaram, K. (2005). *Discrete Particle Swarm Optimization (DPSO) Algorithm for Permutation Flowshop Scheduling to Minimize Makespan*, volume 3612/2005 of *Lecture Notes in Computer Science*, pages 572–581. Springer Berlin / Heidelberg.

[Sato and Yamada, 1995] Sato, A. and Yamada, K. (1995). Generalized learning vector quantization. In Touretzky, D. S., Mozer, M., and Hasselmo, M. E., editors, *NIPS*, pages 423–429. MIT Press.

[Settles and Rylander, 2002] Settles, M. and Rylander, B. (2002). Neural network learning using particle swarm optimizers. *Advances in Information Science and Soft Computing*, pages 1073–1080.

[Shen et al., 2004] Shen, Q., Jiang, J., Jiao, C., Shen, G., and Yu, R. (2004). Modified particle swarm optimization algorithm for variable selection in mlr and pls modeling: Qsar studies of antagonism of angiotensin ii antagonists. *European Journal of Pharmaceutical Sciences*, 22(2-3):145 – 152.

[Shi and Eberhart, 1998] Shi, Y. and Eberhart, R. (1998). A modified particle swarm optimizer. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence, 1998.*, pages 69–73.

[Shi and Eberhart, 1999] Shi, Y. and Eberhart, R. (1999). Empirical study of particle swarm optimization. In *Proceedings of 1999 Congress on Evolutionary Computation, 1999. CEC 99.*, volume 3, pages –1950 Vol. 3.

[Shi et al., 1999] Shi, Y., Eberhart, R., and Chen, Y. (1999). Implementation of evolutionary fuzzy systems. *IEEE Transactions on Fuzzy Systems*, 7(2):109–119.

[Sigaud and Wilson, 2007] Sigaud, O. and Wilson, S. W. (2007). Learning classifier systems: a survey. *Soft Comput.*, 11(11):1065–1078.

[Skalak, 1994] Skalak, D. B. (1994). Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Machine Learning: Proceedings of Eleventh International Conference*, pages 293–301. Morgan Kaufmann.

[Sousa et al., 2004] Sousa, T., Silva, A., and Neves, A. (2004). Particle swarm based data mining algorithms for classification tasks. *Parallel Comput.*, 30(5-6):767–783.

[Suganthan, 1999] Suganthan, P.Ñ. (1999). Particle swarm optimiser with neighbourhood operator. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, pages 1958–1962.

[Tasgetiren et al., 2004] Tasgetiren, M., Sevkli, M., Liang, Y., and Gencyilmaz, G. (2004). Particle swarm optimization algorithm for single machine total weighted tardiness problem. In *Proceedings of IEEE Congress on Evolutionary Computation, 2004. CEC2004.*, volume 2, pages 1412–1419 Vol.2.

[Trelea, 2003] Trelea, I. C. (2003). The particle swarm optimization algorithm: convergence analysis and parameter selection. *Inf. Process. Lett.*, 85(6):317–325.

[Valls et al., 2005] Valls, J., Aler, R., and Fernández, O. (2005). Using a mahalanobis-like distance to train radial basis neural networks. In *8th International Work-Conference on Artificial Neural Networks (IWANN'2005)*, pages 257–263.

[Valls et al., 2007] Valls, J., Aler, R., and Fernández, O. (2007). Evolving generalized euclidean distances for training rbnn. *Computing and Informatics*, 26:33–43.

[van den Bergh and Engelbrecht, 2006] van den Bergh, F. and Engelbrecht, A. (2006). A study of particle swarm optimization particle trajectories. *Information Sciences*, 176(8):937 – 971.

[van der Merwe and Engelbrecht, 2003] van der Merwe, D. and Engelbrecht, A. (2003). Data clustering using particle swarm optimization. In *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, volume 1, pages 215–220 Vol.1.

[Wang et al., 2007] Wang, Z., Sun, X., and Zhan, D. (2007). *A PSO-Based Classification Rule Mining Algorithm*, volume 4682/2007 of *Lecture Notes in Computer Science*, pages 377–384. Springer Berlin / Heidelberg.

[Wang et al., 2006] Wang, Z., Sun, X., and Zhang, D. (2006). *Classification Rule Mining Based on Particle Swarm Optimization*, volume 4062/2006 of *Lecture Notes in Computer Science*, pages 436–441. Springer Berlin / Heidelberg.

[Wettschereck et al., 1997] Wettschereck, D., Aha, D. W., and Mohri, T. (1997). A review and empirical evaluation of feature weighting methods for aclass of lazy learning algorithms. *Artif. Intell. Rev.*, 11(1-5):273–314.

[Wilson and Martinez, 1997] Wilson, D. R. and Martinez, T. R. (1997). Instance pruning techniques. In *ICML '97: Proceedings of Fourteenth Inter-*

national Conference on Machine Learning, pages 403–411, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Wilson and Martinez, 2000] Wilson, D. R. and Martinez, T. R. (2000). Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38(3):257–286.

[Wilson, 1995] Wilson, S. W. (1995). Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175.

[Witten and Frank, 2005] Witten, I. H. and Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco.

[Yuan and Zhao, 2007] Yuan, L. and Zhao, Z. (2007). A modified binary particle swarm optimization algorithm for permutation flow shop problem. In *Proceedings of International Conference on Machine Learning and Cybernetics, 2007*, volume 2, pages 902–907.

[Zhang and Shao, 2000] Zhang, C. and Shao, H. (2000). An ann's evolved by a new evolutionary system and its application. In *Proceedings of 39th IEEE Conference on Decision and Control*, pages 3562–3563.

[Zhang et al., 2008] Zhang, C., Sun, J., Zhu, X., and Yang, Q. (2008). An improved particle swarm optimization algorithm for flowshop scheduling problem. *Inf. Process. Lett.*, 108(4):204–209.

[Zhang et al., 2003] Zhang, L., Zhou, C., Liu, X., Ma, Z., Ma, M., and Liang, Y. (2003). Solving multi objective optimization problems using particle swarm optimization. In *Proceedings of IEEE Congress on Evolutionary Computation 2003 (CEC 2003)*, pages 2400–2405.

[Zhang et al., 2005] Zhang, Q., Li, X., and Tran, Q. (2005). A modified particle swarm optimization algorithm. In *Proceedings of 2005 International Conference on Machine Learning and Cybernetics, 2005.*, volume 5, pages 2993–2995 Vol. 5.

[Zhen et al., 2008] Zhen, L., Wang, L., Wang, X., and Huang, Z. (2008). A novel pso-inspired probability-based binary optimization algorithm. In *Proceedings of International Symposium on Information Science and Engieering, 2008. ISISE '08.*, volume 2, pages 248–251.

[Zheng et al., 2003] Zheng, Y., Ma, L., Zhang, L., and Qian, J. (2003). Empirical study of particle swarm optimizer with an increasing inertia weight. In *Proceedings of The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, volume 1, pages 221–226 Vol.1.

[Zhi et al., 2004] Zhi, X., Xing, X., Wang, Q., Zhang, L., Yang, X., Zhou, C., and Liang, Y. (2004). A discrete pso method for generalized tsp problem. In *Proceedings of 2004 International Conference on Machine Learning and Cybernetics, 2004.*, volume 4, pages 2378–2383 vol.4.